MySQL Exercise 1: Welcome to your first notebook!

Database interfaces vary greatly across platforms and companies. The interface you will be using here, called Jupyter, is a web application designed for data science teams who need to create and share documents that share live code. We will be taking advantage of Jupyter's ability to integrate instructional text with areas that allow you to write and run SQL queries. In this course, you will practice writing queries about the Dognition data set in these Jupyter notebooks as I give you step-by-step instructions through written text. Then once you are comfortable with the query syntax, you will practice writing queries about the Dillard's data set in Teradata Viewpoint's more traditional SQL interface, called SQL scratchpad. Your assessments each week will be based on the exercises you complete using the Dillard's dataset.

Jupyter runs in a language called Python, so some of the commands you will run in these MySQL exercises will require incorporating small amounts of Python code, along with the MySQL query itself. Python is a very popular programming language with many statistical and visualization libraries, so you will likely encounter it in other business analysis settings. Since many data analysis projects do not use Python, however, I will point out to you what parts of the commands are specific to Python interfaces.

The first thing you should do every time you start working with a database: load the SQL library

Since Jupyter is run through Python, the first thing you need to do to start practicing SQL queries is load the SQL library. To do this, type the following line of code into the empty cell below:

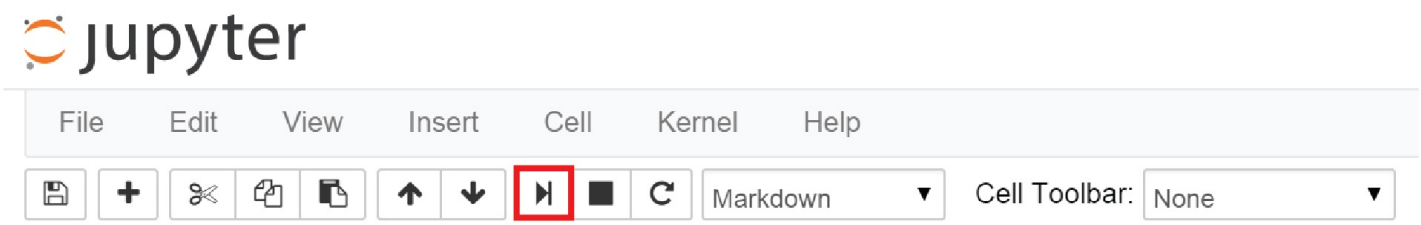
```
%load_ext sql
```

```
In [2]:
```

```
%load_ext sql
```

The "%" in this line of code is syntax for Python, not SQL. The "cell" I am referring to is the empty box area beside the "In [ ]:" you see on the left side of the screen.

Once you've entered the line of code, press the "run" button on the Jupyter toolbar that looks like an arrow. It's located under “cell” in the drop-down menu bar, and next to the stop button (the solid square). The run button is outlined in red in this picture:



TIP: Whenever instructions say “run” or "execute" a command in future exercises, type the appropriate code into the empty cell and execute it by pressing this same button with the arrow.

When the library has loaded, you will see a number between the brackets that preceed the line of code you executed:

For example, it might look like this:

```
In [2]:
```

The second thing you must do every time you want to start working with a database: connect to the database you need to use.

Now that the SQL library is loaded, we need to connect to a database. The following command will log you into the MySQL server at localhost as the user 'studentuser' and will select the database named 'dognitiondb' :

```
mysql://studentuser:studentpw@localhost/dognitiondb
```

However, to execute this command using SQL language instead of Python, you will need to begin the line of code with:

```
%sql
```

Thus, the complete line of code is:

```
%sql mysql://studentuser:studentpw@localhost/dognitiondb
```

Connect to the database by typing this command into the cell below, and running it (note: you can copy and paste the command from above rather than typing it, if you choose):

In [3]:

```
%sql mysql://studentuser:studentpw@localhost/dognitiondb
```

Every time you run a line of SQL code in Jupyter, you will need to preface the line with "%sql". Remember to do this, even though I will not explicitly instruct you to do so for the rest of the exercises in this course.

Once you are connected, the output cell (which reads "Out" followed by brackets) will read: "Connected:studentuser@dognitiondb". To make this the default database for our queries, run this "USE" command:

```
%sql USE dognitiondb
```

In [4]:

```
%sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[4]:

```
[]
```

You are now ready to run queries in the Dognition database!

The third thing you should do every time you start working with a new database: get to know your data

The data sets you will be working with in business settings will be big. REALLY big. If you just start making queries without knowing what you are pulling out, you could hang up your servers or be staring at your computer for hours before you get an output. Therefore, even if you are given an ER diagram or relational schema like we learned about in the first week of the course, before you start querying I strongly recommend that you (1) confirm how many tables each database has, and (2) identify the fields contained in each table of the database. To determine how many tables each database has, use the SHOW command:

```
SHOW tables
```

Try it yourself (TIP: if you get an error message, it's probably because you forgot to start the query with "%sql"):

In [9]:

```
%sql SHOW tables
```

```
 * mysql://studentuser:***@localhost/dognitiondb
6 rows affected.
```

Out[9]:

Tables_in_dognitiondb
complete_tests
dogs
exam_answers
reviews
site_activities
users

The output that appears above should show you there are six tables in the Dognition database. To determine what columns or fields (we will use those terms interchangeably in this course) are in each table, you can use the SHOW command again, but this time (1) you have to clarify that you want to see columns instead of tables, and (2) you have to specify from which table you want to examine the columns.

The syntax, which sounds very similar to what you would actually say in the spoken English language, looks like this:

```
SHOW columns FROM (enter table name here)
```

or if you have multiple databases loaded:

```
SHOW columns FROM (enter table name here) FROM (enter database name here)
```

or

```
SHOW columns FROM databasename.tablename
```

Whenever you have multiple databases loaded, you will need to specify which database a table comes from using one of the syntax options described above.

As I said in the earlier "Introduction to Query Syntax" video, it makes it easier to read and troubleshoot your queries if you always write SQL keywords in UPPERCASE format and write your table and field names in their native format. We will only use the most important SQL keywords in this course, but a full list can be found here:

https://dev.mysql.com/doc/refman/5.7/en/keywords.html (https://dev.mysql.com/doc/refman/5.7/en/keywords.html)

Question 1: How many columns does the "dogs" table have? Enter the appropriate query below to find out:

In [11]:

```
%sql SHOW columns FROM dogs
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[11]:

Field	Type	Null	Key	Default	Extra
gender	varchar(255)	YES		None	
birthday	varchar(255)	YES		None	
breed	varchar(255)	YES		None	
weight	int(11)	YES		None	
dog_fixed	tinyint(1)	YES		None	
dna_tested	tinyint(1)	YES		None	
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
dimension	varchar(255)	YES		None	
exclude	tinyint(1)	YES		None	
breed_type	varchar(255)	YES		None	
breed_group	varchar(255)	YES		None	
dog_guid	varchar(60)	YES	MUL	None	
user_guid	varchar(60)	YES	MUL	None	
total_tests_completed	varchar(255)	YES		None	
mean_iti_days	varchar(255)	YES		None	
mean_iti_minutes	varchar(255)	YES		None	
median_iti_days	varchar(255)	YES		None	
median_iti_minutes	varchar(255)	YES		None	
time_diff_between_first_and_last_game_days	varchar(255)	YES		None	
time_diff_between_first_and_last_game_minutes	varchar(255)	YES		None	

You should have determined that the "dogs" table has 21 columns.

An alternate way to learn the same information would be to use the DESCRIBE function. The syntax is:

```
DESCRIBE tablename
```

Question 2: Try using the DESCRIBE function to learn how many columns are in the "reviews" table:

In [12]:

```
%sql DESCRIBE reviews
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

Out[12]:

Field	Type	Null	Key	Default	Extra
rating	int(11)	YES		None	
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
user_guid	varchar(60)	YES	MUL	None	
dog_guid	varchar(60)	YES	MUL	None	
subcategory_name	varchar(60)	YES		None	
test_name	varchar(60)	YES		None	

You should have determined that there are 7 columns in the "reviews" table.

The SHOW and DESCRIBE functions give a lot more information about the table than just how many columns, or fields, there are. Indeed, the first column of the output shows the title of each field in the table. The column next to that describes what type of data are stored in that column. There are 3 main types of data in MySQL: text, number, and datetime. There are many subtypes of data within these three general categories, as described here:

https://www.hostgator.com/help/article/mysql-variable-types (https://www.hostgator.com/help/article/mysql-variable-types)

The next column in the SHOW/DESCRIBE output indicates whether null values can be stored in the field in the table. The "Key" column of the output provides the following information about each field of data in the table being described (see https://dev.mysql.com/doc/refman/5.7/en/show-columns.html (https://dev.mysql.com/doc/refman/5.7/en/show-columns.html) for more information):

- Empty: the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
- PRI: the column is a PRIMARY KEY or is one of the columns in a multiple-column PRIMARY KEY.
- UNI: the column is the first column of a UNIQUE index.
- MUL: the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

The "Default" field of the output indicates the default value that is assigned to the field. The "Extra" field contains any additional information that is available about a given field in that table.

Questions 3-6: In the cells below, examine the fields in the other 4 tables of the Dognition database:

In [6]:

```
%sql DESCRIBE complete_tests
```

```
 * mysql://studentuser:***@localhost/dognitiondb
6 rows affected.
```

Out[6]:

Field	Type	Null	Key	Default	Extra
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
user_guid	varchar(60)	YES	MUL	None	
dog_guid	varchar(60)	YES	MUL	None	
test_name	varchar(60)	YES		None	
subcategory_name	varchar(60)	YES		None	

In [14]:

```
%sql DESCRIBE exam_answers
```

```
 * mysql://studentuser:***@localhost/dognitiondb
8 rows affected.
```

Out[14]:

Field	Type	Null	Key	Default	Extra
script_detail_id	int(11)	YES		None	
subcategory_name	varchar(255)	YES		None	
test_name	varchar(255)	YES		None	
step_type	varchar(255)	YES		None	
start_time	datetime	YES		None	
end_time	datetime	YES		None	
loop_number	int(11)	YES		None	
dog_guid	varchar(60)	YES		None	

In [5]:

```
%sql DESCRIBE site_activities
```

```
 * mysql://studentuser:***@localhost/dognitiondb
11 rows affected.
```

Out[5]:

Field	Type	Null	Key	Default	Extra
activity_type	varchar(150)	YES	MUL	None	
description	text	YES		None	
membership_id	int(11)	YES		None	
category_id	int(11)	YES		None	
script_id	int(11)	YES		None	
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
user_guid	varchar(255)	YES	MUL	None	
script_detail_id	int(11)	YES		None	
test_name	varchar(255)	YES		None	
dog_guid	varchar(255)	YES	MUL	None	

In [17]:

```
%sql DESCRIBE users
```

```
 * mysql://studentuser:***@localhost/dognitiondb
16 rows affected.
```

Out[17]:

Field	Type	Null	Key	Default	Extra
sign_in_count	int(11)	YES		0	
created_at	datetime	NO		None	
updated_at	datetime	NO		None	
max_dogs	int(11)	YES		0	
membership_id	int(11)	YES		None	
subscribed	tinyint(1)	YES		0	
exclude	tinyint(1)	YES		None	
free_start_user	tinyint(1)	YES		None	
last_active_at	datetime	YES		None	
membership_type	int(11)	YES		None	
user_guid	text	YES	MUL	None	
city	varchar(255)	YES		None	
state	varchar(255)	YES		None	
zip	varchar(255)	YES		None	
country	varchar(255)	YES		None	
utc_correction	varchar(255)	YES		None	

As you examine the fields in each table, you will notice that none of the Dognition tables have primary keys declared. However, take note of which fields say "MUL" in the "Key" column of the DESCRIBE output, because these columns can still be used to link tables together. An important thing to keep in mind, though, is that because these linking columns were not configured as primary keys, it is possible the linking fields contain NULL values or duplicate rows.

If you do not have a ER diagram or relational schema of the Dognition database yet, consider making one at this point, because you will need to refer back to table and column names constantly throughout designing your queries (if you don't remember what primary keys, secondary keys, ER diagrams, or relational schemas are, review the material discussed in the first week of this course). Some database interfaces do provide notes or visual representations about the information in each table for easy reference, but since the Jupyter interface does not provide this, take your own notes and make your own diagrams, and keep them handy. As you will see, these diagrams and notes will save you a lot of time when you design your queries to pull data.

Using SELECT to look at your raw data

Once you have an idea of what is in your tables look like and how they might interact, it's a good idea to look at some of the raw data itself so that you are aware of any anomalies that could pose problems for your analysis or interpretations. To do that, we will use arguably the most important SQL statement for analysts: the SELECT statement.

SELECT is used anytime you want to retrieve data from a table. In order to retrieve that data, you always have to provide at least two pieces of information:

```
(1) what you want to select, and
(2) from where you want to select it.
```

I recommend that you always format your SQL code to ensure that these two pieces of information are on separate lines, so they are easy to identify quickly by eye.

The skeleton of a SELECT statement looks like this:

```
SELECT
FROM
```

To fill in the statement, you indicate the column names you are interested in after "SELECT" and the table name (and database name, if you have multiple databases loaded) you are drawing the information from after "FROM." So in order to look at the breeds in the dogs table, you would execute the following command:

```
SELECT breed
FROM dogs;
```

Remember:

- SQL syntax and keywords are case insensitive. I recommend that you always enter SQL keywords in upper case and table or column names in either lower case or their native format to make it easy to read and troubleshoot your code, but it is not a requirement to do so. Table or column names are often case insensitive as well, but defaults may vary across database platforms so it's always a good idea to check.
- Table or column names with spaces in them need to be surrounded by quotation marks in SQL. MySQL accepts both double and single quotation marks, but some database systems only accept single quotation marks. In all database systems, if a table or column name contains an SQL keyword, the name must be enclosed in backticks instead of quotation marks.

```
'the marks that surrounds this phrase are single quotation marks'
"the marks that surrounds this phrase are double quotation marks"
`the marks that surround this phrase are backticks`
```

- The semi-colon at the end of a query is only required when you have multiple separate queries saved in the same text file or editor. That said, I recommend that you make it a habit to always include a semi-colon at the end of your queries.

An important note for executing queries in Jupyter: in order to tell Python that you want to execute SQL language on multiple lines, you must include two percent signs in front of the SQL prefix instead of one. Therefore, to execute the above query, you should enter:

```
%%sql
SELECT breed
FROM dogs;
```

When Jupyter is busy executing a query, you will see an asterisk in the brackets next to the output field:

```
Out [*]
```

When the query is completed, you will see a number in the brackets next to the output field.

```
Out [5]
```

Try it yourself:

In [32]:

```
%%sql
SELECT breed
FROM dogs LIMIT 5, 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[32]:

breed
Siberian Husky
Shih Tzu
Mixed
Labrador Retriever
Shih Tzu-Poodle Mix
German Shepherd Dog-Pembroke Welsh Corgi Mix
Vizsla
Pug
Boxer
German Shepherd Dog-Nova Scotia Duck Tolling Retriever Mix

I have intentionally limited the output for the purpose of printing my work.

When you do so, you will see a line at the top of the output panel that says "35050 rows affected". This means that there are 35050 rows of data in the dogs table. Each row of the output lists the name of the breed of the dog represented by that entry. Notice that some breed names are listed multiple times, because several dogs of that breed have participated in the Dognition tests.

If you scroll all the way down to the bottom of the output, you will see a notification that says "35050 rows, truncated to displaylimit of 1000." We have set up a display limit of 1000 rows in these notebooks to ensure that our database servers are not overloaded, and to reduce the amount of time you have to wait for the query output. However, in a actual scenario these limits would not necessarily be in place for you. Therefore, before we go any further, I want to show you how you could restrict the number of rows outputted by a query.

CAUTION!!! Every time you select all the data from a column or a table without knowing how much data you are about to pull out, you run the risk of slowing down your network or staring at your screen for hours as you wait for your queries of billions of rows to finish. When in doubt, limit your output. You can always look at more data later.

Using LIMIT to restrict the number of rows in your output (and prevent system crashes)

The MySQL clause you should use is called LIMIT, and it is always placed at the very end of your query. The simplest version of a limit statement looks like this:

```
SELECT breed
FROM dogs LIMIT 5;
```

The "5" in this case indicates that you will only see the first 5 rows of data you select.

Question 7: In the next cell, try entering a query that will let you see the first 10 rows of the breed column in the dogs table.

In [7]:

```
%%sql
SELECT breed
FROM dogs
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[7]:

breed
Labrador Retriever
Shetland Sheepdog
Golden Retriever
Golden Retriever
Shih Tzu
Siberian Husky
Shih Tzu
Mixed
Labrador Retriever
Shih Tzu-Poodle Mix

You can also select rows of data from different parts of the output table, rather than always just starting at the beginning. To do this, use the OFFSET clause after LIMIT. The number after the OFFSET clause indicates from which row the output will begin querying. Note that the offset of Row 1 of a table is actually 0. Therefore, in the following query:

```
SELECT breed
FROM dogs LIMIT 10 OFFSET 5;
```

10 rows of data will be returned, starting at Row 6.

An alternative way to write the OFFSET clause in the query is:

```
SELECT breed
FROM dogs LIMIT 5, 10;
```

In this notation, the offset is the number before the comma, and the number of rows returned is the number after the comma.

Try it yourself:

In [8]:

```
%%sql
SELECT breed
FROM dogs
LIMIT 10 OFFSET 5;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[8]:

breed
Siberian Husky
Shih Tzu
Mixed
Labrador Retriever
Shih Tzu-Poodle Mix
German Shepherd Dog-Pembroke Welsh Corgi Mix
Vizsla
Pug
Boxer
German Shepherd Dog-Nova Scotia Duck Tolling Retriever Mix

The LIMIT command is one of the pieces of syntax that can vary across database platforms. MySQL uses LIMIT to restrict the output, but other databases including Teradata use a statement called "TOP" instead. Oracle has yet another syntax:

http://www.tutorialspoint.com/sql/sql-top-clause.htm (http://www.tutorialspoint.com/sql/sql-top-clause.htm)

Make sure to look up the correct syntax for the database type you are using.

Using SELECT to query multiple columns

Now that we know how to limit our output, we are ready to make our SELECT statement work a little harder. The SELECT statement can be used to select multiple columns as well as a single column. The output of the query will depend on the order of the columns you enter after the SELECT statement in your query. When you enter column names, separate each name with a comma, but do NOT include a comma after the last column name.

Try the following query with different orders of the column names to observe the differences in output (I will include a LIMIT statement in many of the examples I use in the rest of the course, but feel free to change or remove them to explore different aspects of the data):

```
SELECT breed, breed_type, breed_group
FROM dogs LIMIT 5, 10;
```

In [11]:

```
%%sql
SELECT breed, breed_type, breed_group
FROM dogs
LIMIT 5, 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[11]:

breed	breed_type	breed_group
Siberian Husky	Pure Breed	Working
Shih Tzu	Pure Breed	Toy
Mixed	Mixed Breed/ Other/ I Don't Know	None
Labrador Retriever	Pure Breed	Sporting
Shih Tzu-Poodle Mix	Cross Breed	None
German Shepherd Dog-Pembroke Welsh Corgi Mix	Cross Breed	None
Vizsla	Pure Breed	Sporting
Pug	Pure Breed	Toy
Boxer	Pure Breed	Working
German Shepherd Dog-Nova Scotia Duck Tolling Retriever Mix	Cross Breed	None

Another trick to know about when using SELECT is that you can use an asterisk as a "wild card" to return all the data in a table. (A wild card is defined as a character that will represent or match any character or sequence of characters in a query.) Take note, this is very risky to do if you do not limit your output or if you don't know how many data are in your database, so use the wild card with caution. However, it is a handy tool to use when you don't have all the column names easily available or when you know you want to query an entire table.

The syntax is as follows:

```
SELECT *
FROM dogs LIMIT 5, 10;
```

Question 8: Try using the wild card to query the reviews table:

In [13]:

```
%%sql
SELECT *
FROM reviews
LIMIT 5, 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[13]:

rating	created_at	updated_at	user_guid	dog_guid	subcategory_name	test_name
6	2014-05-01 22:38:33	2014-05-01 22:38:33	ce47172c-7144-11e5-ba71-058fbc01cf0b	ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Game
3	2014-05-01 22:45:24	2014-05-01 22:45:24	ce47172c-7144-11e5-ba71-058fbc01cf0b	ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Game
0	2014-05-02 01:32:49	2014-05-02 01:32:49	ce471eca-7144-11e5-ba71-058fbc01cf0b	ce405e28-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm-up
9	2014-05-02 01:52:06	2014-05-02 01:52:06	ce471eca-7144-11e5-ba71-058fbc01cf0b	ce405e28-7144-11e5-ba71-058fbc01cf0b	Cunning	Turn Your Back
0	2014-05-02 02:58:23	2014-05-02 02:58:23	ce261aa4-7144-11e5-ba71-058fbc01cf0b	ce2609c4-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm-up
6	2014-05-02 03:05:29	2014-05-02 03:05:29	ce261aa4-7144-11e5-ba71-058fbc01cf0b	ce2609c4-7144-11e5-ba71-058fbc01cf0b	Communication	Arm Pointing
2	2014-05-02 03:12:47	2014-05-02 03:12:47	ce261aa4-7144-11e5-ba71-058fbc01cf0b	ce2609c4-7144-11e5-ba71-058fbc01cf0b	Communication	Foot Pointing
7	2014-05-02 11:50:27	2014-05-02 11:50:27	ce261bd0-7144-11e5-ba71-058fbc01cf0b	ce260c1c-7144-11e5-ba71-058fbc01cf0b	Shaker Game	Shaker Warm-Up
0	2014-05-02 18:48:36	2014-05-02 18:48:36	ce2ab050-7144-11e5-ba71-058fbc01cf0b	ce3fd48a-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Game
0	2014-05-02 18:52:07	2014-05-02 18:52:07	ce2ab050-7144-11e5-ba71-058fbc01cf0b	ce3fd48a-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Warm-up

NOTE: if you do this for the dogs table, your output will be too wide to see at one time in the dialog box. Use the scroll bars at bottom and to the right of the dialog box to see the entire output table.

SELECT statements can also be used to make new derivations of individual columns using "+" for addition, "-" for subtraction, "*" for multiplication, or "/" for division. For example, if you wanted the median inter-test intervals in hours instead of minutes or days, you could query:

```
SELECT median_iti_minutes / 60
FROM dogs LIMIT 5, 10;
```

Question 9: Go ahead and try it, adding in a column to your output that shows you the original median_iti in minutes.

In [14]:

```
%%sql
SELECT median_iti_minutes / 60, median_iti_minutes
FROM dogs
LIMIT 5, 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[14]:

median_iti_minutes / 60	median_iti_minutes
0.085555555285	5.1333333171
0.0080555537245	0.48333322347
0.11138889105833334	6.6833334635
0.081111109755	4.8666665853
0.09333333570666666	5.6000001424
0.14027777682833334	8.4166666097
0.10750000088166667	6.4500000529
0.09583333319833334	5.7499999919
0.094444443495	5.6666666097
0.09055555474166667	5.4333332845

Now it's time to practice writing your own SELECT statements.

Question 10: How would you retrieve the first 15 rows of data from the dog_guid, subcategory_name, and test_name fields of the Reviews table, in that order?

In [15]:

```
%%sql
SELECT dog_guid, subcategory_name, test_name
FROM reviews
LIMIT 15;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
15 rows affected.
```

Out[15]:

dog_guid	subcategory_name	test_name
ce3ac77e-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Warm-up
ce2aedcc-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Warm-up
ce2aedcc-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Game
ce2aedcc-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm-up
ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Warm-up
ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Game
ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Game
ce405e28-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm-up
ce405e28-7144-11e5-ba71-058fbc01cf0b	Cunning	Turn Your Back
ce2609c4-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm-up
ce2609c4-7144-11e5-ba71-058fbc01cf0b	Communication	Arm Pointing
ce2609c4-7144-11e5-ba71-058fbc01cf0b	Communication	Foot Pointing
ce260c1c-7144-11e5-ba71-058fbc01cf0b	Shaker Game	Shaker Warm-Up
ce3fd48a-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Game
ce3fd48a-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Warm-up

Question 11: How would you retrieve 10 rows of data from the activity_type, created_at, and updated_at fields of the site_activities table, starting at row 50? What do you notice about the created_at and updated_at fields?

In [16]:

```
%%sql
SELECT activity_type, created_at, Updated_at
FROM site_activities
LIMIT 49, 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[16]:

activity_type	created_at	Updated_at
point_in_cat	2013-07-25 20:55:08	2013-07-25 20:55:08
point_in_cat	2013-07-25 20:55:07	2013-07-25 20:55:07
point_in_cat	2013-07-25 20:55:50	2013-07-25 20:55:50
point_in_cat	2013-07-25 20:56:09	2013-07-25 20:56:09
point_in_cat	2013-07-25 20:56:21	2013-07-25 20:56:21
point_in_cat	2013-07-25 21:00:31	2013-07-25 21:00:31
point_in_cat	2013-07-25 21:02:29	2013-07-25 21:02:29
point_in_cat	2013-07-25 21:02:31	2013-07-25 21:02:31
point_in_cat	2013-07-25 21:02:45	2013-07-25 21:02:45
point_in_cat	2013-07-25 21:05:41	2013-07-25 21:05:41

Question 12: How would you retrieve 20 rows of data from all the columns in the users table, starting from row 2000? What do you notice about the free_start_user field?

In [17]:

```
%%sql
SELECT *
FROM users
LIMIT 20 OFFSET 1999;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[17]:

sign_in_count	created_at	updated_at	max_dogs	membership_id	subscribed	exclude	free_start_user	last_active_at	membership_type
1	2013-02-21 23:19:08	2015-01-28 20:51:53	1	1	1	None	None	None	1
3	2013-02-21 23:25:03	2015-01-28 20:51:53	1	2	1	None	None	None	2
1	2013-02-22 01:21:20	2015-01-28 20:51:53	1	1	1	None	None	None	1
1	2013-02-22 03:05:42	2015-01-28 20:51:53	1	2	1	None	None	None	2
5	2013-02-22 03:29:06	2015-01-28 20:51:53	1	1	1	None	None	2014-06-23 23:31:28	1
1	2013-02-22 15:05:13	2015-01-28 20:51:53	1	1	1	None	None	None	1
1	2013-02-22 16:30:17	2015-01-28 20:51:53	1	2	1	None	None	None	2
1	2013-02-22 18:16:05	2015-01-28 20:51:53	1	1	0	None	None	None	1
3	2013-02-22 20:26:53	2015-01-28 20:51:53	1	1	1	None	None	None	1
1	2013-02-22 23:14:40	2015-01-28 20:51:54	2	2	1	None	None	None	2
1	2013-02-23 13:05:38	2015-01-28 20:51:54	1	1	1	None	None	None	1
3	2013-02-23 16:56:55	2015-01-28 20:51:54	2	2	1	None	None	None	2
3	2013-02-23 21:07:41	2015-01-28 20:51:54	1	2	1	None	None	None	2
1	2013-02-23 21:22:09	2015-01-28 20:51:54	1	1	1	None	None	None	1
4	2013-02-23 22:48:15	2015-01-28 20:51:54	1	2	1	None	None	2014-08-04 14:16:34	2
4	2013-02-24 00:14:21	2015-01-28 20:51:54	2	2	0	None	None	None	2
1	2013-02-24 07:34:32	2015-01-28 20:51:54	1	1	0	None	None	None	1
	2013-02-	2015-01-28							

1	24 14:48:43	20:51:54	1	1	1	None	None	None	1
1	2013-02-24 15:34:10	2015-01-28 20:51:54	1	1	1	None	None	None	1
1	2013-02-24 15:36:19	2015-01-28 20:51:54	1	1	0	None	None	None	1

You have already learned how to see all the data in your database! Congratulations!

Feel free to practice any other queries you are interested in below:

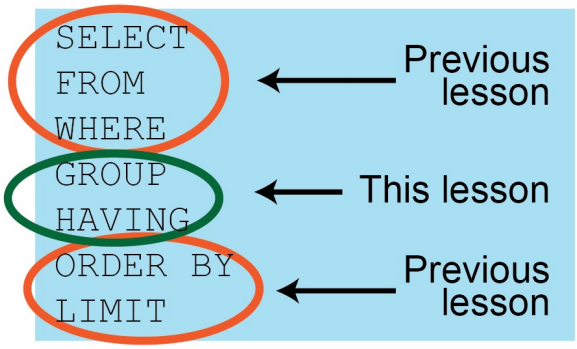
In [ ]:

MySQL Exercise 5: Summaries of Groups of Data

So far you've learned how to select, reformat, manipulate, order, and summarize data from a single table in database. In this lesson, you are going to learn how to summarize multiple subsets of your data in the same query. The method for doing this is to include a "GROUP BY" clause in your SQL query.

The GROUP BY clause

The GROUP BY clause comes after the WHERE clause, but before ORDER BY or LIMIT:



The GROUP BY clause is easy to incorporate into your queries. In fact, it might be a little too easy to incorporate into MySQL queries, because it can be used incorrectly in MySQL queries even when no error message is displayed. As a consequence, I suggest you adopt a healthy dose of caution every time you use the GROUP BY clause. By the end of this lesson, you will understand why. When used correctly, though, GROUP BY is one of the most useful and efficient parts of an SQL query, and once you are comfortable using it, you will use it very frequently.

To get started, load the SQL library and the Dognition database, and set the dognition database as the default:

In [39]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[39]:

```
[]
```

Let's return to a question from MySQL Exercise 4. How would you query the average rating for each of the 40 tests in the Reviews table? As we discussed, one very inefficient method to do that would be to write 40 separate queries with each one having a different test name in the WHERE conditional clause. Then you could copy or transcribe the results from all 40 queries into one place. But that wouldn't be very pleasant. Here's how you could do the same thing using one query that has a GROUP BY clause: ```mySQL SELECT test_name, AVG(rating) AS AVG_Rating FROM reviews GROUP BY test_name ``` This query will output the average rating for each test. More technically, this query will instruct MySQL to average all the rows that have the same value in the test_name column. Notice that I included test_name in the SELECT statement. As a strong rule of thumb, if you are grouping by a column, you should also include that column in the SELECT statement. If you don't do this, you won't know to which group each row of your output corresponds. **To see what I mean, try the query above without test_name included in the SELECT statement:**

In [5]:

```
%%sql
SELECT test_name, AVG(rating) AS AVG_Rating
FROM reviews
GROUP BY test_name;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
40 rows affected.
```

Out[5]:

test_name	AVG_Rating
1 vs 1 Game	3.9206
3 vs 1 Game	4.2857
5 vs 1 Game	3.9272
Arm Pointing	4.2153
Cover Your Eyes	2.6741
Delayed Cup Game	3.3514
Different Perspective	2.7647
Expression Game	4.0000
Eye Contact Game	2.9372
Eye Contact Warm-up	0.9632
Foot Pointing	4.0093
Impossible Task Game	3.0965
Impossible Task Warm-up	0.2174
Inferential Reasoning Game	4.5223
Inferential Reasoning Warm-up	4.3066
Memory versus Pointing	3.5584
Memory versus Smell	4.2623
Navigation Game	2.9841
Navigation Learning	2.0303
Navigation Warm-up	1.9805
Numerosity Warm-Up	2.6173
One Cup Warm-up	1.3693
Physical Reasoning Game	3.8492
Physical Reasoning Warm-up	1.6625
Self Control Game	3.8519
Shaker Game	4.6667
Shaker Warm-Up	2.1818
Shared Perspective	3.2778
Slide	4.5111
Smell Game	4.2857
Stair Game	4.2857
Switch	5.5676
Treat Warm-up	0.7909
Turn Your Back	3.1293
Two Cup Warm-up	1.6737
Warm-Up	1.2020
Watching	2.4594
Watching - Part 2	2.6570
Yawn Game	2.8477
Yawn Warm-up	2.0035

You can form groups using derived values as well as original columns. To illustrate this, let's address another question: how many tests were completed during each month of the year?

To answer this question, we need to take advantage of another datetime function described in the website below:

http://www.w3resource.com/mysql/date-and-time-functions/date-and-time-functions.php (http://www.w3resource.com/mysql/date-and-time-functions/date-and-time-functions.php)

MONTH() will return a number representing the month of a date entry. To get the total number of tests completed each month, you could put the MONTH function into the GROUP BY clause, in this case through an alias:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY Month;
```

You can also group by multiple columns or derived fields. If we wanted to determine the total number of each type of test completed each month, you could include both "test_name" and the derived "Month" field in the GROUP BY clause, separated by a comma.

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY test_name, Month;
```

MySQL allows you to use aliases in a GROUP BY clause, but some database systems do not. If you are using a database system that does NOT accept aliases in GROUP BY clauses, you can still group by derived fields, but you have to duplicate the calculation for the derived field in the GROUP BY clause in addition to including the derived field in the SELECT clause:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY test_name, MONTH(created_at);
```

Try the query once with test_name first in the GROUP BY list, and once with Month first in the GROUP BY list below. Inspect the outputs:

In [7]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS NUM_Completed_Tests
FROM complete_tests
GROUP BY MOnth;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
12 rows affected.
```

Out[7]:

test_name	Month	NUM_Completed_Tests
Delayed Cup Game	1	11068
Yawn Warm-up	2	9122
Yawn Warm-up	3	9572
Physical Reasoning Game	4	7130
Delayed Cup Game	5	21013
Foot Pointing	6	23381
Eye Contact Game	7	15977
Memory versus Smell	8	13382
Yawn Warm-up	9	19853
Yawn Warm-up	10	39237
Inferential Reasoning Warm-up	11	12652
Inferential Reasoning Warm-up	12	10859

In [11]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS NUM_Completed_Tests
FROM complete_tests
GROUP BY test_name, MOnth
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[11]:

test_name	Month	NUM_Completed_Tests
1 vs 1 Game	1	25
1 vs 1 Game	2	28
1 vs 1 Game	3	22
1 vs 1 Game	4	12
1 vs 1 Game	5	13
1 vs 1 Game	6	18
1 vs 1 Game	7	36
1 vs 1 Game	8	17
1 vs 1 Game	9	28
1 vs 1 Game	10	27
1 vs 1 Game	11	15
1 vs 1 Game	12	14
3 vs 1 Game	1	35
3 vs 1 Game	2	28
3 vs 1 Game	3	34
3 vs 1 Game	4	16
3 vs 1 Game	5	34
3 vs 1 Game	6	42
3 vs 1 Game	7	37
3 vs 1 Game	8	23

In [10]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS NUM_Completed_Tests
FROM complete_tests
GROUP BY test_name, MOnth(created_at)
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[10]:

test_name	Month	NUM_Completed_Tests
1 vs 1 Game	1	25
1 vs 1 Game	2	28
1 vs 1 Game	3	22
1 vs 1 Game	4	12
1 vs 1 Game	5	13
1 vs 1 Game	6	18
1 vs 1 Game	7	36
1 vs 1 Game	8	17
1 vs 1 Game	9	28
1 vs 1 Game	10	27
1 vs 1 Game	11	15
1 vs 1 Game	12	14
3 vs 1 Game	1	35
3 vs 1 Game	2	28
3 vs 1 Game	3	34
3 vs 1 Game	4	16
3 vs 1 Game	5	34
3 vs 1 Game	6	42
3 vs 1 Game	7	37
3 vs 1 Game	8	23

In [12]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS NUM_Completed_Tests
FROM complete_tests
GROUP BY MOnth(created_at), test_name
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

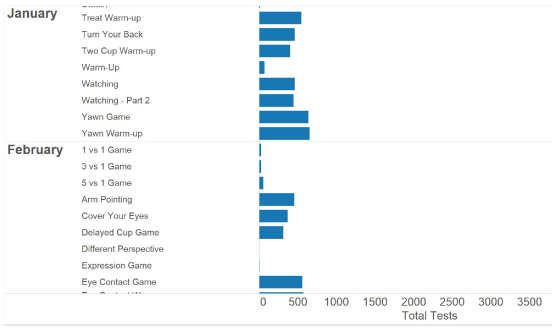
Out[12]:

test_name	Month	NUM_Completed_Tests
1 vs 1 Game	1	25
3 vs 1 Game	1	35
5 vs 1 Game	1	59
Arm Pointing	1	622
Cover Your Eyes	1	452
Delayed Cup Game	1	356
Different Perspective	1	2
Expression Game	1	6
Eye Contact Game	1	624
Eye Contact Warm-up	1	707
Foot Pointing	1	506
Impossible Task Game	1	49
Impossible Task Warm-up	1	51
Inferential Reasoning Game	1	410
Inferential Reasoning Warm-up	1	425
Memory versus Pointing	1	464
Memory versus Smell	1	465
Navigation Game	1	57
Navigation Learning	1	59
Navigation Warm-up	1	62

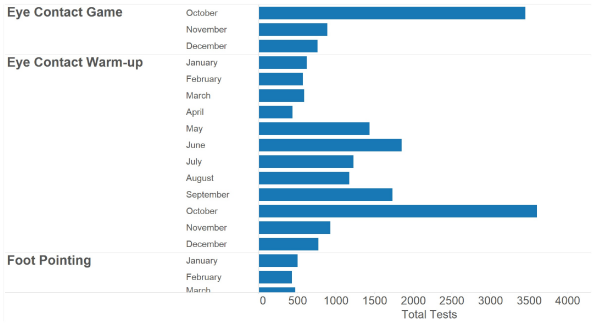
Notice that in the first case, the first block of rows share the same test_name, but are broken up into separate months (for those of you who took the "Data Visualization and Communication with Tableau" course of this specialization, this is similar to what would happen if you put test_name first and created_at second on the rows or columns shelf in Tableau).

In the second case, the first block of rows share the same month, but are broken up into separate tests (this is similar to what would happen if you put created_at first and test_name second on the rows or columns shelf in Tableau). If you were to visualize these outputs, they would look like the charts below.

GROUP BY MONTH(created_at), test_name;



GROUP BY test_name, MONTH(created_at)



Different database servers might default to ordering the outputs in a certain way, but you shouldn't rely on that being the case. To ensure the output is ordered in a way you intend, add an ORDER BY clause to your grouped query using the syntax you already know and have practiced:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY test_name, Month
ORDER BY test_name ASC, Month ASC;
```

Question 1: Output a table that calculates the number of distinct female and male dogs in each breed group of the Dogs table, sorted by the total number of dogs in descending order (the sex/breed_group pair with the greatest number of dogs should have 8466 unique Dog_Guids):

In [14]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS NUM_Completed_Tests
FROM complete_tests
GROUP BY test_name, MOnth
ORDER BY test_name ASC, Month ASC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[14]:

test_name	Month	NUM_Completed_Tests
1 vs 1 Game	1	25
1 vs 1 Game	2	28
1 vs 1 Game	3	22
1 vs 1 Game	4	12
1 vs 1 Game	5	13
1 vs 1 Game	6	18
1 vs 1 Game	7	36
1 vs 1 Game	8	17
1 vs 1 Game	9	28
1 vs 1 Game	10	27
1 vs 1 Game	11	15
1 vs 1 Game	12	14
3 vs 1 Game	1	35
3 vs 1 Game	2	28
3 vs 1 Game	3	34
3 vs 1 Game	4	16
3 vs 1 Game	5	34
3 vs 1 Game	6	42
3 vs 1 Game	7	37
3 vs 1 Game	8	23

Question 1 Answer

In [15]:

```
%%sql
SELECT gender, breed_group, COUNT(DISTINCT dog_guid) AS Num_Dogs
FROM dogs
GROUP BY gender, breed_group
ORDER BY Num_Dogs DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
18 rows affected.
```

Out[15]:

gender	breed_group	Num_Dogs
male	None	8466
female	None	8367
male	Sporting	2584
female	Sporting	2262
male	Herding	1736
female	Herding	1704
male	Toy	1473
female	Toy	1145
male	Non-Sporting	1098
male	Working	1075
female	Non-Sporting	919
male	Terrier	919
female	Working	895
female	Terrier	794
male	Hound	725
female	Hound	614
male		147
female		127

Some database servers, including MySQL, allow you to use numbers in place of field names in the GROUP BY or ORDER BY fields to reduce the overall length of the queries. I tend to avoid this abbreviated method of writing queries because I find it challenging to troubleshoot when you are writing complicated queries with many fields, but it does allow you to write queries faster. To use this method, assign each field in your SELECT statement a number acording to the order the field appears in the SELECT statement. In the following statement:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
```

test_name would be #1, Month would be #2, and Num_Completed_Tests would be #3. You could then rewrite the query above to read:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY 1, 2
ORDER BY 1 ASC, 2 ASC;
```

Question 2: Revise the query your wrote in Question 1 so that it uses only numbers in the GROUP BY and ORDER BY fields.

In [16]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS NUM_Completed_Tests
FROM complete_tests
GROUP BY 1, 2
ORDER BY 1 ASC, 2 ASC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[16]:

test_name	Month	NUM_Completed_Tests
1 vs 1 Game	1	25
1 vs 1 Game	2	28
1 vs 1 Game	3	22
1 vs 1 Game	4	12
1 vs 1 Game	5	13
1 vs 1 Game	6	18
1 vs 1 Game	7	36
1 vs 1 Game	8	17
1 vs 1 Game	9	28
1 vs 1 Game	10	27
1 vs 1 Game	11	15
1 vs 1 Game	12	14
3 vs 1 Game	1	35
3 vs 1 Game	2	28
3 vs 1 Game	3	34
3 vs 1 Game	4	16
3 vs 1 Game	5	34
3 vs 1 Game	6	42
3 vs 1 Game	7	37
3 vs 1 Game	8	23

Questions 2 Answer

In [18]:

```
%%sql
SELECT gender, breed_group, COUNT(DISTINCT dog_guid) AS Num_Dogs
FROM dogs
GROUP BY 1, 2
ORDER BY 3 DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
18 rows affected.
```

Out[18]:

gender	breed_group	Num_Dogs
male	None	8466
female	None	8367
male	Sporting	2584
female	Sporting	2262
male	Herding	1736
female	Herding	1704
male	Toy	1473
female	Toy	1145
male	Non-Sporting	1098
male	Working	1075
female	Non-Sporting	919
male	Terrier	919
female	Working	895
female	Terrier	794
male	Hound	725
female	Hound	614
male		147
female		127

The HAVING clause

Just like you can query subsets of rows using the WHERE clause, you can query subsets of aggregated groups using the HAVING clause. However, wheras the expression that follows a WHERE clause has to be applicable to each row of data in a column, the expression that follows a HAVING clause has to be applicable or computable using a group of data.

If you wanted to examine the number of tests completed only during the winter holiday months of November and December, you would need to use a WHERE clause, because the month a test was completed in is recorded in each row. Your query might look like this:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
WHERE MONTH(created_at)=11 OR MONTH(created_at)=12
GROUP BY 1, 2
ORDER BY 3 DESC;
```

If you then wanted to output only the test-month pairs that had at least 20 records in them, you would add a HAVING clause, because the stipulation of at least 20 records only makes sense and is only computable at the aggregated group level:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
WHERE MONTH(created_at)=11 OR MONTH(created_at)=12
GROUP BY 1, 2
HAVING COUNT(created_at)>=20
ORDER BY 3 DESC;
```

Question 3: Revise the query your wrote in Question 2 so that it (1) excludes the NULL and empty string entries in the breed_group field, and (2) excludes any groups that don't have at least 1,000 distinct Dog_Guids in them. Your result should contain 8 rows. (HINT: sometimes empty strings are registered as non-NULL values. You might want to include the following line somewhere in your query to exclude these values as well):

```
breed_group!=""
```

In [30]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
WHERE  MONTH(created_at)=11 OR MONTH(created_at)=12
GROUP BY 1,2
ORDER BY 3 DESC
LIMIT 40, 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[30]:

test_name	Month	Num_Completed_Tests
Warm-Up	11	65
5 vs 1 Game	12	59
Warm-up	12	56
Navigation Warm-up	11	52
Navigation Warm-up	12	49
Impossible Task Warm-up	11	49
Impossible Task Game	11	48
Navigation Game	11	47
Navigation Learning	11	47
Navigation Learning	12	47
Numerosity Warm-Up	12	46
Impossible Task Game	12	46
Impossible Task Warm-up	12	46
Navigation Game	12	45
3 vs 1 Game	12	45
5 vs 1 Game	11	40
Numerosity Warm-Up	11	26
3 vs 1 Game	11	22
1 vs 1 Game	11	15
Self Control Game	11	15

In [25]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
WHERE  MONTH(created_at)=11 AND MONTH(created_at)=12
GROUP BY 1, 2
ORDER BY 3 DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[25]:

test_name	Month	Num_Completed_Tests

In [27]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
WHERE  MONTH(created_at)=11 OR MONTH(created_at)=12
GROUP BY 1, 2
HAVING COUNT(created_at)>20
ORDER BY 3 DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
58 rows affected.
```

Out[27]:

test_name	Month	Num_Completed_Tests
Yawn Warm-up	11	1060
Yawn Warm-up	12	1043

Yawn Game	11	989
Yawn Game	12	978
Eye Contact Warm-up	11	941
Eye Contact Game	11	889
Eye Contact Warm-up	12	839
Eye Contact Game	12	760
Treat Warm-up	11	744
Arm Pointing	11	735
Foot Pointing	11	685
Treat Warm-up	12	667
Arm Pointing	12	666
Watching	11	611
Turn Your Back	11	601
Cover Your Eyes	11	595
Foot Pointing	12	566
Watching - Part 2	11	565
Watching	12	555
One Cup Warm-up	11	491
Two Cup Warm-up	11	482
Memory versus Pointing	11	477
Turn Your Back	12	467
Cover Your Eyes	12	447
Memory versus Smell	11	444
Watching - Part 2	12	438
Delayed Cup Game	11	422
One Cup Warm-up	12	401
Two Cup Warm-up	12	379
Inferential Reasoning Warm-up	11	376
Memory versus Pointing	12	376
Inferential Reasoning Game	11	362
Memory versus Smell	12	352
Physical Reasoning Warm-up	11	349
Physical Reasoning Game	11	341
Delayed Cup Game	12	299
Inferential Reasoning Warm-up	12	296
Inferential Reasoning Game	12	280
Physical Reasoning Warm-up	12	276
Physical Reasoning Game	12	250
Warm-Up	11	65
5 vs 1 Game	12	59
Warm-up	12	56
Navigation Warm-up	11	52
Impossible Task Warm-up	11	49
Navigation Warm-up	12	49
Impossible Task Game	11	48
Navigation Game	11	47
Navigation Learning	11	47
Navigation Learning	12	47
Numerosity Warm-Up	12	46
Impossible Task Game	12	46
Impossible Task Warm-up	12	46
Navigation Game	12	45
3 vs 1 Game	12	45
5 vs 1 Game	11	40
Numerosity Warm-Up	11	26

3 vs 1 Game	11	22

Question 3 solution

In [31]:

```
%%sql
SELECT gender, breed_group, COUNT(DISTINCT dog_guid) AS Num_Dogs
FROM dogs
WHERE breed_group IS NOT NULL AND breed_group!="None" AND breed_group!=""
GROUP BY 1, 2
HAVING Num_Dogs>1000
ORDER BY 3 DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
8 rows affected.
```

Out[31]:

gender	breed_group	Num_Dogs
male	Sporting	2584
female	Sporting	2262
male	Herding	1736
female	Herding	1704
male	Toy	1473
female	Toy	1145
male	Non-Sporting	1098
male	Working	1075

We will review several issues that can be tricky about using GROUP BY in your queries in the next lesson, but those issues will make more sense once you are sure you are comfortable with the basic functionality of the GROUP BY and HAVING clauses.

Practice incorporating GROUP BY and HAVING into your own queries.

Question 4: Write a query that outputs the average number of tests completed and average mean inter-test-interval for every breed type, sorted by the average number of completed tests in descending order (popular hybrid should be the first row in your output).

In [33]:

```
%%sql
SELECT breed_type, AVG(total_tests_completed) AS TotTests, AVG(mean_iti_minutes) AS AvgMeaniti
FROM dogs
GROUP BY breed_type
ORDER BY AVG(total_tests_completed) DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[33]:

breed_type	TotTests	AvgMeaniti
Popular Hybrid	10.257530120481928	2834.3205728931534
Cross Breed	9.945900537634408	2872.351156110182
Pure Breed	9.871602824737856	3193.350493795222
Mixed Breed/ Other/ I Don't Know	9.54250850170034	3023.0711302156274

Question 5: Write a query that outputs the average amount of time it took customers to complete each type of test where any individual reaction times over 6000 hours are excluded and only average reaction times that are greater than 0 seconds are included (your output should end up with 67 rows).

In [37]:

```
%%sql
SELECT test_name, AVG(TIMESTAMPDIFF(HOUR,start_time,end_time)) AS Duration
FROM exam_answers
WHERE TIMESTAMPDIFF(HOUR,start_time,end_time)<6000
GROUP BY test_name
HAVING AVG(TIMESTAMPDIFF(SECOND,start_time,end_time))>0
ORDER BY Duration DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
67 rows affected.
```

Out[37]:

test_name	Duration
Social-Quiz	81.9923
Diet	32.7712
Sociability	21.1726
Activity	20.0071
Yawn Warm-up	15.2669
Shy/Boldness	11.6170
Watching	11.4743
Attachment	11.0557
Excitability	10.9972
Surprise And Delight	10.9355
Confinement	9.2003
Set 3	7.0007
Navigation Warm-up	6.1733
Gender	6.1326
Physical	5.6950
Owner	5.2280
Emotions	4.6607
Numerosity Warm-Up	4.5937
Yawn Game	3.8349
Recall	3.8206
Perception	3.8185
Impossible Task Warm-up	3.8022
Puzzles	3.7676
Partnership	3.4735
Obedience	3.4643
Purina-Only	3.3696
Set 1	3.0899
Social	2.7286
Training	2.6089
Purina	2.4655
Treat Warm-up	2.2185
Toys	2.2110
Watching - Part 2	2.1875
Warm-Up	2.0840
Delayed Cup Game	1.4267
One Cup Warm-up	1.4191
Inferential Reasoning Warm-up	1.3832
Eye Contact Warm-up	1.2267
Memory versus Smell	1.1865
Smell Game	1.1813
Eye Contact Game	1.1389
Cover Your Eyes	0.8247
Turn Your Back	0.7646
Arm Pointing	0.6092
Foot Pointing	0.5739
Slide	0.5362
Environment	0.4971
Physical Reasoning Warm-up	0.4921
Inferential Reasoning Game	0.3883
Stair Game	0.3021
Navigation Learning	0.2271
Memory versus Pointing	0.2205
Impossible Task Game	0.1913

Navigation Game	0.1273
Set 2	0.1259
Two Cup Warm-up	0.1223
Physical Reasoning Game	0.0819
Shaker Warm-Up	0.0644
Switch	0.0531
Different Perspective	0.0493
Shaker Game	0.0000
Shared Perspective	0.0000
1 vs 1 Game	0.0000
Self Control Game	0.0000
5 vs 1 Game	0.0000
Expression Game	0.0000
3 vs 1 Game	0.0000

Question 6: Write a query that outputs the total number of unique User_Guids in each combination of State and ZIP code (postal code) in the United States, sorted first by state name in ascending alphabetical order, and second by total number of unique User_Guids in descending order (your first state should be AE and there should be 5043 rows in total in your output).

In [42]:

```
%%sql
SELECT state, zip, COUNT(DISTINCT user_guid) AS Num_Users
FROM users
WHERE Country='US'
GROUP BY State, zip
ORDER BY State ASC, Num_Users DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[42]:

state	zip	Num_Users
AE	9128	2
AE	9845	1
AE	9053	1
AE	9107	1
AE	9469	1
AK	99507	3
AK	99709	3
AK	99501	2
AK	99577	2
AK	99502	1
AK	99509	1
AK	99516	1
AK	99518	1
AK	99567	1
AK	99587	1
AK	99611	1
AK	99645	1
AK	99676	1
AK	99705	1
AK	99712	1

Question 7: Write a query that outputs the total number of unique User_Guids in each combination of State and ZIP code in the United States *that have at least 5 users*, sorted first by state name in ascending alphabetical order, and second by total number of unique User_Guids in descending order (your first state/ZIP code combination should be AZ/86303).

In [44]:

```
%%sql
SELECT state, zip, COUNT(DISTINCT user_guid) AS Num_Users
FROM users
WHERE Country='US'
GROUP BY State, zip
HAVING Num_Users >=5
ORDER BY State ASC, Num_Users DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[44]:

state	zip	Num_Users
AZ	86303	14
AZ	85718	6
AZ	85749	5
AZ	85253	5
AZ	85254	5
AZ	85260	5
AZ	85711	5
CA	92107	16
CA	90046	13
CA	94107	12
CA	92130	12
CA	92024	10
CA	94110	10
CA	94611	9
CA	94114	9
CA	94025	9
CA	94941	9
CA	90049	8
CA	92064	8
CA	90068	8

Be sure to watch the next video before beginning Exercise 6, the next set of MySQL exercises , and feel free to practice any other queries you wish below!

In [ ]:

```

```

MySQL Exercise 9: Subqueries and Derived Tables

Now that you understand how joins work, in this lesson we are going to learn how to incorporate subqueries and derived tables into our queries.

Subqueries, which are also sometimes called inner queries or nested queries, are queries that are embedded within the context of another query. The output of a subquery is incorporated into the queries that surround it. Subqueries can be used in SELECT, WHERE, and FROM clauses. When they are used in FROM clauses they create what are called derived tables.

The main reasons to use subqueries are:

- Sometimes they are the most logical way to retrieve the information you want
- They can be used to isolate each logical part of a statement, which can be helpful for troubleshooting long and complicated queries
- Sometimes they run faster than joins

Some people find subqueries easier to read than joins. However, that is often a result of not feeling comfortable with the concepts behind joins in the first place (I prefer join syntax, so admittedly, that is my preference).

Subqueries must be enclosed in parentheses. Subqueries have a couple of rules that joins don't:

- ORDER BY phrases cannot be used in subqueries (although ORDER BY phrases can still be used in outer queries that contain subqueries).
- Subqueries in SELECT or WHERE clauses that return more than one row must be used in combination with operators that are explicitly designed to handle multiple values, such as the IN operator. Otherwise, subqueries in SELECT or WHERE statements can output no more than 1 row.

So why would you use subqueries?

Let's look at some examples.

Start by loading the sql library and database, and making the Dognition database your default database:

In [2]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[2]:

```
[]
```

1) "On the fly calculations" (or, doing calculations as you need them)

One of the main uses of subqueries is to calculate values as you need them. This allows you to use a summary calculation in your query without having to enter the value outputted by the calculation explicitly. A situation when this capability would be useful is if you wanted to see all the records that were greater than the average value of a subset of your data.

Recall one of the queries we wrote in "MySQL Exercise 4: Summarizing your Data" to calculate the average amount of time it took customers to complete all of the tests in the exam_answers table (we had to exclude negative durations from the calculation due to some abnormalities in the data):

```
SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time)>0;
```

What if we wanted to look at just the data from rows whose durations were greater than the average, so that we could determine whether there are any features that seem to correlate with dogs taking a longer time to finish their tests? We could use a subquery to calculate the average duration, and then indicate in our SELECT and WHERE clauses that we only wanted to retrieve the rows whose durations were greater than the average. Here's what the query would look like:

```
SELECT *
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time) >
    (SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
     FROM exam_answers
     WHERE TIMESTAMPDIFF(minute,start_time,end_time)>0);
```

You can see that TIMESTAMPDIFF gets compared to the singular average value outputted by the subquery surrounded by parentheses. You can also see that it's easier to read the query as a whole if you indent and align all the clauses associated with the subquery, relative to the main query.

Question 1: How could you use a subquery to extract all the data from exam_answers that had test durations that were greater than the average duration for the "Yawn Warm-Up" game? Start by writing the query that gives you the average duration for the "Yawn Warm-Up" game by itself (and don't forget to exclude negative values; your average duration should be about 9934):

In [4]:

```
%%sql
SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time)>0;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[4]:

AvgDuration
11233.0951

In [3]:

```
%%sql
SELECT *
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time) >
    (SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
     FROM exam_answers
     WHERE TIMESTAMPDIFF(minute,start_time,end_time)>0)
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[3]:

script_detail_id	subcategory_name	test_name	step_type	start_time	end_time	loop_number	dog_guid
537	Sociability	Sociability	question	2013-02-05 03:58:13	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
538	Emotions	Emotions	question	2013-02-05 03:58:31	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
539	Shy/Boldness	Shy/Boldness	question	2013-02-05 03:59:03	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
540	Perception	Perception	question	2013-02-05 03:59:10	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
541	Recall	Recall	question	2013-02-05 03:59:22	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
542	Attachment	Attachment	question	2013-02-05 03:59:36	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
543	Puzzles	Puzzles	question	2013-02-05 03:59:41	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
544	Shy/Boldness	Shy/Boldness	question	2013-02-05 04:00:00	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
545	Shy/Boldness	Shy/Boldness	question	2013-02-05 04:00:16	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
546	Partnership	Partnership	question	2013-02-05 04:00:35	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b

CODE

%%sql SELECT * FROM exam_answers WHERE TIMESTAMPDIFF(minute,start_time,end_time) > (SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration FROM exam_answers WHERE TIMESTAMPDIFF(minute,start_time,end_time)>0);

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 10572 rows affected.

QUESTION 1 ANSWER

In [7]:

```
%%sql
SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time)>0 AND test_name="Yawn Warm-Up";
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[7]:

AvgDuration
9933.5197

Question 2: Once you've verified that your subquery is written correctly on its own, incorporate it into a main query to extract all the data from exam_answers that had test durations that were greater than the average duration for the "Yawn Warm-Up" game (you will get 11059 rows):

In [2]:

```
%%sql
SELECT *
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time) >
(SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time)>0 AND test_name="Yawn Warm-Up")
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[2]:

script_detail_id	subcategory_name	test_name	step_type	start_time	end_time	loop_number	dog_guid
537	Sociability	Sociability	question	2013-02-05 03:58:13	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
538	Emotions	Emotions	question	2013-02-05 03:58:31	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
539	Shy/Boldness	Shy/Boldness	question	2013-02-05 03:59:03	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
540	Perception	Perception	question	2013-02-05 03:59:10	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
541	Recall	Recall	question	2013-02-05 03:59:22	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
542	Attachment	Attachment	question	2013-02-05 03:59:36	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
543	Puzzles	Puzzles	question	2013-02-05 03:59:41	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
544	Shy/Boldness	Shy/Boldness	question	2013-02-05 04:00:00	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
545	Shy/Boldness	Shy/Boldness	question	2013-02-05 04:00:16	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
546	Partnership	Partnership	question	2013-02-05 04:00:35	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
547	Emotions	Emotions	question	2013-02-05 04:00:46	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
548	Perception	Perception	question	2013-02-05 04:00:54	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
549	Obedience	Obedience	question	2013-02-05 04:01:01	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
550	Attachment	Attachment	question	2013-02-05 04:01:15	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
551	Attachment	Attachment	question	2013-02-05 04:01:40	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
552	Puzzles	Puzzles	question	2013-02-05 04:02:02	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
553	Recall	Recall	question	2013-02-05 04:02:30	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
554	Obedience	Obedience	question	2013-02-05 04:03:00	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
555	Perception	Perception	question	2013-02-05 04:03:29	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
556	Sociability	Sociability	question	2013-02-05 04:03:37	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b

In [3]:

```
%%sql
SELECT *
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time) > "9934"
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[3]:

script_detail_id	subcategory_name	test_name	step_type	start_time	end_time	loop_number	dog_guid
537	Sociability	Sociability	question	2013-02-05 03:58:13	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
538	Emotions	Emotions	question	2013-02-05 03:58:31	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
539	Shy/Boldness	Shy/Boldness	question	2013-02-05 03:59:03	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
540	Perception	Perception	question	2013-02-05 03:59:10	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
541	Recall	Recall	question	2013-02-05 03:59:22	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
542	Attachment	Attachment	question	2013-02-05 03:59:36	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
543	Puzzles	Puzzles	question	2013-02-05 03:59:41	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
544	Shy/Boldness	Shy/Boldness	question	2013-02-05 04:00:00	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
545	Shy/Boldness	Shy/Boldness	question	2013-02-05 04:00:16	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
546	Partnership	Partnership	question	2013-02-05 04:00:35	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
547	Emotions	Emotions	question	2013-02-05 04:00:46	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
548	Perception	Perception	question	2013-02-05 04:00:54	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
549	Obedience	Obedience	question	2013-02-05 04:01:01	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
550	Attachment	Attachment	question	2013-02-05 04:01:15	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
551	Attachment	Attachment	question	2013-02-05 04:01:40	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
552	Puzzles	Puzzles	question	2013-02-05 04:02:02	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
553	Recall	Recall	question	2013-02-05 04:02:30	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
554	Obedience	Obedience	question	2013-02-05 04:03:00	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
555	Perception	Perception	question	2013-02-05 04:03:29	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b
556	Sociability	Sociability	question	2013-02-05 04:03:37	2013-10-02 20:18:06	0	fd27b272-7144-11e5-ba71-058fbc01cf0b

Now double check the results you just retrieved by replacing the subquery with "9934"; you should get the same results. It is helpful to get into the habit of including these kinds of quality checks into your query-writing process.

This example shows you how subqueries allow you retrieve information dynamically, rather than having to hard code in specific numbers or names. This capability is particularly useful when you need to build the output of your queries into reports or dashboards that are supposed to display real-time information.

2) Testing membership

Subqueries can also be useful for assessing whether groups of rows are members of other groups of rows. To use them in this capacity, we need to know about and practice the IN, NOT IN, EXISTS, and NOT EXISTS operators.

Recall from MySQL Exercise 2: Selecting Data Subsets Using WHERE that the IN operator allows you to use a WHERE clause to say how you want your results to relate to a list of multiple values. It's basically a condensed way of writing a sequence of OR statements. The following query would select all the users who live in the state of North Carolina (abbreviated "NC") or New York (abbreviated "NY"):

```
SELECT *
FROM users
WHERE state IN ('NC','NY');
```

Notice the quotation marks around the members of the list referred to by the IN statement. These quotation marks are required since the state names are strings of text.

A query that would give an equivalent result would be:

```
SELECT *
FROM users
WHERE state ='NC' OR state ='NY';
```

A query that would select all the users who do NOT live in the state of North Carolina or New York would be:

```
SELECT *
FROM users
WHERE state NOT IN ('NC','NY');
```

Question 3: Use an IN operator to determine how many entries in the exam_answers tables are from the "Puzzles", "Numerosity", or "Bark Game" tests. You should get a count of 163022.

In [4]:

```
%%sql
SELECT *
FROM users
WHERE state IN ('NC','NY')
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[4]:

sign_in_count	created_at	updated_at	max_dogs	membership_id	subscribed	exclude	free_start_user	last_active_at	membership_type
181	2013-02-05 17:54:42	2015-01-28 20:51:49	13	2	1	1	0	None	2
181	2013-02-05 17:54:42	2015-01-28 20:51:49	13	2	1	1	0	None	2
65	2013-02-05 00:52:16	2015-01-28 20:51:49	3	2	1	None	None	None	2
7	2013-02-06 00:40:59	2015-01-28 20:51:50	1	2	1	None	None	None	2
15	2013-02-06 14:13:42	2015-01-28 20:51:50	1	2	1	None	None	None	2
181	2013-02-05 17:54:42	2015-01-28 20:51:49	13	2	1	1	0	None	2
2	2013-02-06 19:50:16	2015-01-28 20:51:50	2	2	1	None	None	None	2
181	2013-02-05 17:54:42	2015-01-28 20:51:49	13	2	1	1	0	None	2
4	2013-02-07 04:19:57	2015-01-28 20:51:50	2	1	0	None	None	2014-06-23 20:42:36	1
2	2013-02-06 01:54:46	2015-01-28 20:51:50	1	1	0	None	None	None	1

In [ ]:

```
CODE

%%sql
SELECT *
FROM users
WHERE state IN ('NC','NY');

OUTPUT

* mysql://studentuser:***@localhost/dognitiondb
1333 rows affected.
```

In [6]:

```
%%sql
SELECT *
FROM users
WHERE state NOT IN ('NC','NY')
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[6]:

sign_in_count	created_at	updated_at	max_dogs	membership_id	subscribed	exclude	free_start_user	last_active_at	membership_type
14	2013-02-05 03:52:02	2015-03-12 00:25:15	2	2	1	None	None	2015-03-12 00:25:15	2
8	2013-02-05 15:29:50	2015-01-28 20:51:49	2	1	1	None	None	None	1
3	2013-02-05 17:21:18	2015-01-28 20:51:49	1	1	1	None	None	None	1
21	2013-02-05 13:50:48	2015-01-28 20:51:49	1	1	1	None	None	2014-09-12 19:26:29	1
6	2013-02-05 18:02:03	2015-01-28 20:51:49	1	1	1	None	None	None	1
6	2013-02-05 17:58:23	2015-01-28 20:51:49	1	1	0	None	None	None	1
13	2013-02-05 20:55:34	2015-01-28 20:51:50	1	1	1	0	0	None	1
4	2013-02-05 21:27:19	2015-01-28 20:51:50	1	2	0	None	None	None	2
7	2013-02-05 22:13:07	2015-01-28 20:51:50	2	2	1	None	None	None	2
10	2013-02-05 23:05:03	2015-01-28 20:51:50	1	2	0	None	None	None	2

In [9]:

```
%%sql
SELECT COUNT(*)
FROM users
WHERE state NOT IN ('NC','NY');
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[9]:

COUNT(*)
16652

QUESTION 3 ANSWER

In [8]:

```
%%sql
SELECT COUNT(*)
FROM exam_answers
WHERE subcategory_name IN ('Puzzles','Numerosity','Bark Game');
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[8]:

COUNT(*)
163022

Question 4: Use a NOT IN operator to determine how many unique dogs in the dog table are NOT in the "Working", "Sporting", or "Herding" breeding groups. You should get an answer of 7961.

In [10]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM dogs
WHERE breed_group NOT IN ('Working','Sporting','Herding');
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[10]:

COUNT(DISTINCT dog_guid)
7961

EXISTS and NOT EXISTS perform similar functions to IN and NOT IN, but EXISTS and NOT EXISTS can only be used in subqueries. The syntax for EXISTS and NOT EXISTS statements is a little different than that of IN statements because EXISTS is not preceded by a column name or any other expression. The most important difference between EXISTS/NOT EXISTS and IN/NOT IN statements, though, is that unlike IN/NOT IN statements, EXISTS/NOT EXISTS are logical statements. Rather than returning raw data, per se, EXISTS/NOT EXISTS statements return a value of TRUE or FALSE. As a practical consequence, EXISTS statements are often written using an asterisk after the SELECT clause rather than explicit column names. The asterisk is faster to write, and since the output is just going to be a logical true/false either way, it does not matter whether you use an asterisk or explicit column names.

We can use EXISTS and a subquery to compare the users who are in the users table and dogs table, similar to what we practiced previously using joins. If we wanted to retrieve a list of all the users in the users table who were also in the dogs table, we could write:

```
SELECT DISTINCT u.user_guid AS uUserID
FROM users u
WHERE EXISTS (SELECT d.user_guid
              FROM dogs d
              WHERE u.user_guid =d.user_guid);
```

You would get the same result if you wrote:

```
SELECT DISTINCT u.user_guid AS uUserID
FROM users u
WHERE EXISTS (SELECT *
              FROM dogs d
              WHERE u.user_guid =d.user_guid);
```

Essentially, both of these queries say give me all the distinct user_guids from the users table that have a value of "TRUE" in my EXISTS clause. The results would be equivalent to an inner join with GROUP BY query. Now...

Question 5: How could you determine the number of unique users in the users table who were NOT in the dogs table using a NOT EXISTS clause? You should get the 2226, the same result as you got in Question 10 of MySQL Exercise 8: Joining Tables with Outer Joins.

In [14]:

```
%%sql
SELECT DISTINCT u.user_guid AS uUserID
FROM users u
WHERE EXISTS (SELECT d.user_guid
              FROM dogs d
              WHERE u.user_guid =d.user_guid)
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[14]:

uUserID
ce134e42-7144-11e5-ba71-058fbc01cf0b
ce1353d8-7144-11e5-ba71-058fbc01cf0b
ce135ab8-7144-11e5-ba71-058fbc01cf0b
ce13507c-7144-11e5-ba71-058fbc01cf0b
ce135e14-7144-11e5-ba71-058fbc01cf0b
ce13615c-7144-11e5-ba71-058fbc01cf0b
ce135f2c-7144-11e5-ba71-058fbc01cf0b
ce136a1c-7144-11e5-ba71-058fbc01cf0b
ce136ac6-7144-11e5-ba71-058fbc01cf0b
ce136c24-7144-11e5-ba71-058fbc01cf0b

In [13]:

```
%%sql
SELECT COUNT(DISTINCT u.user_guid) AS uUserID
FROM users u
WHERE EXISTS (SELECT d.user_guid
              FROM dogs d
              WHERE u.user_guid =d.user_guid);
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[13]:

uUserID
30967

In [15]:

```
%%sql
SELECT DISTINCT u.user_guid AS uUserID
FROM users u
WHERE EXISTS (SELECT *
              FROM dogs d
              WHERE u.user_guid =d.user_guid)
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[15]:

uUserID
ce134e42-7144-11e5-ba71-058fbc01cf0b
ce1353d8-7144-11e5-ba71-058fbc01cf0b
ce135ab8-7144-11e5-ba71-058fbc01cf0b
ce13507c-7144-11e5-ba71-058fbc01cf0b
ce135e14-7144-11e5-ba71-058fbc01cf0b
ce13615c-7144-11e5-ba71-058fbc01cf0b
ce135f2c-7144-11e5-ba71-058fbc01cf0b
ce136a1c-7144-11e5-ba71-058fbc01cf0b
ce136ac6-7144-11e5-ba71-058fbc01cf0b
ce136c24-7144-11e5-ba71-058fbc01cf0b

QUESTION 5 ANSWER

In [17]:

```
%%sql
SELECT DISTINCT u.user_guid AS uUserID
FROM users u
WHERE NOT EXISTS (SELECT d.user_guid
FROM dogs d
WHERE u.user_guid =d.user_guid)
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[17]:

uUserID
ce134f50-7144-11e5-ba71-058fbc01cf0b
ce135888-7144-11e5-ba71-058fbc01cf0b
ce1359aa-7144-11e5-ba71-058fbc01cf0b
ce135fea-7144-11e5-ba71-058fbc01cf0b
ce1360a8-7144-11e5-ba71-058fbc01cf0b
ce13642c-7144-11e5-ba71-058fbc01cf0b
ce1364d6-7144-11e5-ba71-058fbc01cf0b
ce13663e-7144-11e5-ba71-058fbc01cf0b
ce136792-7144-11e5-ba71-058fbc01cf0b
ce136832-7144-11e5-ba71-058fbc01cf0b

In [ ]:

```
CODE

%%sql
SELECT DISTINCT u.user_guid AS uUserID
FROM users u
WHERE NOT EXISTS (SELECT d.user_guid
FROM dogs d
WHERE u.user_guid =d.user_guid);

OUTPUT

* mysql://studentuser:***@localhost/dognitiondb
2226 rows affected.
```

3) Accurate logical representations of desired output and Derived Tables

A third situation in which subqueries can be useful is when they simply represent the logic of what you want better than joins.

We saw an example of this in our last MySQL Exercise. We wanted a list of each dog a user in the users table owns, with its accompanying breed information whenever possible. To achieve this, we wrote this query in Question 6:

```
SELECT u.user_guid AS uUserID, d.user_guid AS dUserID, d.dog_guid AS dDogID, d.breed
FROM users u LEFT JOIN dogs d
   ON u.user_guid=d.user_guid
```

Once we saw the "exploding rows" phenomenon due to duplicate rows, we wrote a follow-up query in Question 7 to assess how many rows would be outputted per user_id when we left joined the users table on the dogs table:

```
SELECT u.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
FROM users u LEFT JOIN dogs d
   ON u.user_guid=d.user_guid
GROUP BY u.user_guid
ORDER BY numrows DESC
```

This same general query without the COUNT function could have been used to output a complete list of all the distinct users in the users table, their dogs, and their dogs' breed information. However, the method we used to arrive at this was not very pretty or logically satisfying. Rather than joining many duplicated rows and fixing the results later with the GROUP BY clause, it would be much more elegant if we could simply join the distinct UserIDs in the first place. There is no way to do that with join syntax, on its own. However, you can use subqueries in combination with joins to achieve this goal.

To complete the join on ONLY distinct UserIDs from the users table, we could write:

```
SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid
      FROM users u) AS DistinctUUsersID
LEFT JOIN dogs d
   ON DistinctUUsersID.user_guid=d.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC
```

Try it yourself:

In [19]:

```
%%sql
SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid
      FROM users u) AS DistinctUUsersID LEFT JOIN dogs d
   ON DistinctUUsersID.user_guid=d.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[19]:

uUserID	dUserID	numrows
ce7b75bc-7144-11e5-ba71-058fbc01cf0b	ce7b75bc-7144-11e5-ba71-058fbc01cf0b	1819
ce225842-7144-11e5-ba71-058fbc01cf0b	ce225842-7144-11e5-ba71-058fbc01cf0b	26
ce2258a6-7144-11e5-ba71-058fbc01cf0b	ce2258a6-7144-11e5-ba71-058fbc01cf0b	20
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	13
ce29675e-7144-11e5-ba71-058fbc01cf0b	ce29675e-7144-11e5-ba71-058fbc01cf0b	11
ce134492-7144-11e5-ba71-058fbc01cf0b	ce134492-7144-11e5-ba71-058fbc01cf0b	9
ce6676d0-7144-11e5-ba71-058fbc01cf0b	ce6676d0-7144-11e5-ba71-058fbc01cf0b	8
ce83d2ca-7144-11e5-ba71-058fbc01cf0b	ce83d2ca-7144-11e5-ba71-058fbc01cf0b	8
ce32305a-7144-11e5-ba71-058fbc01cf0b	ce32305a-7144-11e5-ba71-058fbc01cf0b	7
ce7adeea-7144-11e5-ba71-058fbc01cf0b	ce7adeea-7144-11e5-ba71-058fbc01cf0b	7

CODE

%%sql SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows FROM (SELECT DISTINCT u.user_guid FROM users u) AS DistinctUUsersID LEFT JOIN dogs d ON DistinctUUsersID.user_guid=d.user_guid GROUP BY DistinctUUsersID.user_guid ORDER BY numrows DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 33193 rows affected.

Queries that include subqueries always run the innermost subquery first, and then run subsequent queries sequentially in order from the innermost query to the outermost query.

Therefore, the query we just wrote extracts the distinct user_guids from the users table *first*, and then left joins that reduced subset of user_guids on the dogs table. As mentioned at the beginning of the lesson, since the subquery is in the FROM statement, it actually creates a temporary table, called a derived table, that is then incorporated into the rest of the query.

There are several important points to notice about the syntax of this subquery. First, an alias of "DistinctUUsersID" is used to name the results of the subquery. *We are required to give an alias to any derived table we create in subqueries within FROM statements.* Otherwise there would be no way for the database to refer to the multiple columns within the temporary results we create.

Second, *we need to use this alias every time we want to execute a function that uses the derived table.* Remember that the results in which we are interested require a join between the dogs table and the temporary table, not the dogs table and the original users table with duplicates. That means we need to make sure we reference the temporary table alias in the ON, GROUP BY, and SELECT clauses.

Third, relatedly, aliases used within subqueries can refer to tables outside of the subqueries. However, *outer queries cannot refer to aliases created within subqueries unless those aliases are explicitly part of the subquery output*. In other words, if you wrote the first line of the query above as:

```
SELECT u.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
...
```

the query would not execute because the alias "u" is contained inside the subquery, but is not included in the output. **Go ahead and try it to see what the error message looks like:**

In [ ]:

```
%%sql
SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid
      FROM users u) AS DistinctUUsersID LEFT JOIN dogs d
   ON DistinctUUsersID.user_guid=d.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC;
```

A similar thing would happen if you tried to use the alias u in the GROUP BY statement.

Another thing to take note of is that when you use subqueries in FROM statements, the temporary table you create can have multiple columns in the output (unlike when you use subqueries in outside SELECT statements). But for that same reason, subqueries in FROM statements can be very computationally intensive. Therefore, it's a good idea to use them sparingly, especially when you have very large data sets.

Overall, subqueries and joins can often be used interchangeably. Some people strongly prefer one approach over another, but there is no consensus about which approach is best. When you are analyzing very large datasets, it's a good idea to test which approach will likely be faster or easier to troubleshoot for your particular application.

Let's practice some more subqueries!

Question 6: Write a query using an IN clause and equijoin syntax that outputs the dog_guid, breed group, state of the owner, and zip of the owner for each distinct dog in the Working, Sporting, and Herding breed groups. (You should get 10,254 rows; the query will be a little slower than some of the others we have practiced)

In [4]:

```
%%sql
SELECT DISTINCT d.dog_guid, d.breed_group, u.state, u.zip
FROM dogs d, users u
WHERE breed_group IN ('Working','Sporting','Herding') AND d.user_guid=u.user_guid
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[4]:

dog_guid	breed_group	state	zip
fd27b272-7144-11e5-ba71-058fbc01cf0b	Sporting	ND	58201
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	Herding	MA	1005
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b	Herding	MA	1005
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	Sporting	CT	6820
fd27b79a-7144-11e5-ba71-058fbc01cf0b	Sporting	IL	60093
fd27b948-7144-11e5-ba71-058fbc01cf0b	Working	WA	98001
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	Sporting	WA	98117
fd27c0fa-7144-11e5-ba71-058fbc01cf0b	Sporting	WA	98117
fd27c7d0-7144-11e5-ba71-058fbc01cf0b	Sporting	CA	95003
fd27c8d4-7144-11e5-ba71-058fbc01cf0b	Working	VA	22903

CODE

%%sql SELECT DISTINCT d.dog_guid, d.breed_group, u.state, u.zip FROM dogs d, users u WHERE breed_group IN ('Working','Sporting','Herding') AND d.user_guid=u.user_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 10254 rows affected.

Question 7: Write the same query as in Question 6 using traditional join syntax.

In [6]:

```
%%sql
SELECT DISTINCT d.dog_guid, d.breed_group, u.state, u.zip
FROM dogs d JOIN users u
ON d.user_guid=u.user_guid
WHERE breed_group IN ('Working','Sporting','Herding')
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[6]:

dog_guid	breed_group	state	zip
fd27b272-7144-11e5-ba71-058fbc01cf0b	Sporting	ND	58201
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	Herding	MA	1005
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b	Herding	MA	1005
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	Sporting	CT	6820
fd27b79a-7144-11e5-ba71-058fbc01cf0b	Sporting	IL	60093
fd27b948-7144-11e5-ba71-058fbc01cf0b	Working	WA	98001
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	Sporting	WA	98117
fd27c0fa-7144-11e5-ba71-058fbc01cf0b	Sporting	WA	98117
fd27c7d0-7144-11e5-ba71-058fbc01cf0b	Sporting	CA	95003
fd27c8d4-7144-11e5-ba71-058fbc01cf0b	Working	VA	22903

CODE

%%sql SELECT DISTINCT d.dog_guid, d.breed_group, u.state, u.zip FROM dogs d JOIN users u ON d.user_guid=u.user_guid WHERE breed_group IN ('Working','Sporting','Herding');

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 10254 rows affected.

Question 8: Earlier we examined unique users in the users table who were NOT in the dogs table. Use a NOT EXISTS clause to examine all the users in the dogs table that are not in the users table (you should get 2 rows in your output).

In [7]:

```
%%sql
SELECT d.user_guid AS dUserID, d.dog_guid AS dDogID
FROM dogs d
WHERE NOT EXISTS (SELECT DISTINCT u.user_guid
FROM users u
WHERE d.user_guid =u.user_guid);
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[7]:

dUserID	dDogID
None	fd7c0a66-7144-11e5-ba71-058fbc01cf0b
None	fdbb6b7a-7144-11e5-ba71-058fbc01cf0b

Question 9: We saw earlier that user_guid 'ce7b75bc-7144-11e5-ba71-058fbc01cf0b' still ends up with 1819 rows of output after a left outer join with the dogs table. If you investigate why, you'll find out that's because there are duplicate user_guids in the dogs table as well. How would you adapt the query we wrote earlier (copied below) to only join unique UserIDs from the users table with unique UserIDs from the dog table?

Join we wrote earlier:

```
SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid
      FROM users u) AS DistinctUUsersID
LEFT JOIN dogs d
  ON DistinctUUsersID.user_guid=d.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC;
```

Let's build our way up to the correct query. To troubleshoot, let's only examine the rows related to user_guid 'ce7b75bc-7144-11e5-ba71-058fbc01cf0b', since that's the userID that is causing most of the trouble. Rewrite the query above to only LEFT JOIN *distinct* user(s) from the user table whose user_guid='ce7b75bc-7144-11e5-ba71-058fbc01cf0b'. The first two output columns should have matching user_guids, and the numrows column should have one row with a value of 1819:

In [10]:

```
%%sql
SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid
      FROM users u) AS DistinctUUsersID
LEFT JOIN dogs d
  ON DistinctUUsersID.user_guid=d.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[10]:

uUserID	dUserID	numrows
ce7b75bc-7144-11e5-ba71-058fbc01cf0b	ce7b75bc-7144-11e5-ba71-058fbc01cf0b	1819
ce225842-7144-11e5-ba71-058fbc01cf0b	ce225842-7144-11e5-ba71-058fbc01cf0b	26
ce2258a6-7144-11e5-ba71-058fbc01cf0b	ce2258a6-7144-11e5-ba71-058fbc01cf0b	20
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	13
ce29675e-7144-11e5-ba71-058fbc01cf0b	ce29675e-7144-11e5-ba71-058fbc01cf0b	11
ce134492-7144-11e5-ba71-058fbc01cf0b	ce134492-7144-11e5-ba71-058fbc01cf0b	9
ce6676d0-7144-11e5-ba71-058fbc01cf0b	ce6676d0-7144-11e5-ba71-058fbc01cf0b	8
ce83d2ca-7144-11e5-ba71-058fbc01cf0b	ce83d2ca-7144-11e5-ba71-058fbc01cf0b	8
ce32305a-7144-11e5-ba71-058fbc01cf0b	ce32305a-7144-11e5-ba71-058fbc01cf0b	7
ce7adeea-7144-11e5-ba71-058fbc01cf0b	ce7adeea-7144-11e5-ba71-058fbc01cf0b	7

CODE

%%sql SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows FROM (SELECT DISTINCT u.user_guid FROM users u) AS DistinctUUsersID LEFT JOIN dogs d ON DistinctUUsersID.user_guid=d.user_guid GROUP BY DistinctUUsersID.user_guid ORDER BY numrows DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 33193 rows affected.

In [11]:

```
%%sql
SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS
numrows
FROM (SELECT DISTINCT u.user_guid
FROM users u
WHERE u.user_guid='ce7b75bc-7144-11e5-ba71-058fbc01cf0b') AS
DistinctUUsersID
LEFT JOIN dogs d
ON DistinctUUsersID.user_guid=d.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[11]:

uUserID	dUserID	numrows
ce7b75bc-7144-11e5-ba71-058fbc01cf0b	ce7b75bc-7144-11e5-ba71-058fbc01cf0b	1819

Question 10: Now let's prepare and test the inner query for the right half of the join. Give the dogs table an alias, and write a query that would select the distinct user_guids from the dogs table (we will use this query as a inner subquery in subsequent questions, so you will need an alias to differentiate the user_guid column of the dogs table from the user_guid column of the users table).

In [19]:

```
%%sql
SELECT DISTINCT d.user_guid
FROM dogs d
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[19]:

user_guid
None
ce134492-7144-11e5-ba71-058fbc01cf0b
ce134a78-7144-11e5-ba71-058fbc01cf0b
ce134be0-7144-11e5-ba71-058fbc01cf0b
ce134d16-7144-11e5-ba71-058fbc01cf0b
ce134e42-7144-11e5-ba71-058fbc01cf0b
ce13507c-7144-11e5-ba71-058fbc01cf0b
ce135194-7144-11e5-ba71-058fbc01cf0b
ce1352ac-7144-11e5-ba71-058fbc01cf0b
ce1353d8-7144-11e5-ba71-058fbc01cf0b

CODE

%%sql SELECT DISTINCT d.user_guid FROM dogs d;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 30968 rows affected.

CODE

%%sql SELECT DISTINCT d.user_guid FROM dogs d;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 30968 rows affected.

Question 11: Now insert the query you wrote in Question 10 as a subquery on the right part of the join you wrote in question 9. The output should return columns that should have matching user_guids, and 1 row in the numrows column with a value of 1. If you are getting errors, make sure you have given an alias to the derived table you made to extract the distinct user_guids from the dogs table, and double-check that your aliases are referenced correctly in the SELECT and ON statements.

In [13]:

```
%%sql
SELECT DistinctUUsersID.user_guid AS uUserID, DistictDUsersID.user_guid AS
dUserID, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid
FROM users u
WHERE u.user_guid='ce7b75bc-7144-11e5-ba71-058fbc01cf0b') AS
DistinctUUsersID
LEFT JOIN (SELECT DISTINCT d.user_guid
FROM dogs d) AS DistictDUsersID
ON DistinctUUsersID.user_guid=DistictDUsersID.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[13]:

uUserID	dUserID	numrows
ce7b75bc-7144-11e5-ba71-058fbc01cf0b	ce7b75bc-7144-11e5-ba71-058fbc01cf0b	1

Question 12: Adapt the query from Question 10 so that, in theory, you would retrieve a full list of all the DogIDs a user in the users table owns, with its accompagnying breed information whenever possible. HOWEVER, BEFORE YOU RUN THE QUERY MAKE SURE TO LIMIT YOUR OUTPUT TO 100 ROWS *WITHIN* THE SUBQUERY TO THE LEFT OF YOUR JOIN. If you run the query without imposing limits it will take a *very* long time. If you try to limit the output by just putting a limit clause at the end of the outermost query, the database will still have to hold the entire derived tables in memory and join each row of the derived tables before limiting the output. If you put the limit clause in the subquery to the left of the join, the database will only have to join 100 rows of data.

In [16]:

```
%%sql
SELECT DistinctUUsersID.user_guid AS uUserID, DistictDUsersID.user_guid AS
dUserID,
DistictDUsersID.dog_guid AS DogID, DistictDUsersID.breed AS breed
FROM (SELECT DISTINCT u.user_guid
FROM users u
LIMIT 100) AS DistinctUUsersID
LEFT JOIN (SELECT DISTINCT d.user_guid, d.dog_guid, d.breed
FROM dogs d) AS DistictDUsersID
ON DistinctUUsersID.user_guid=DistictDUsersID.user_guid
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[16]:

uUserID	dUserID	DogID	breed
ce134e42-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	fd27b272-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever
ce134e42-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	fd417cac-7144-11e5-ba71-058fbc01cf0b	Mixed
ce1353d8-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog
ce1353d8-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	fd3fb0f2-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog
ce135ab8-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce13507c-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b	fd27b79a-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27e9a4-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ed46-7144-11e5-ba71-058fbc01cf0b	Shih Tzu

CODE

%%sql SELECT DistinctUUsersID.user_guid AS uUserID, DistictDUsersID.user_guid AS dUserID, DistictDUsersID.dog_guid AS DogID, DistictDUsersID.breed AS breed FROM (SELECT DISTINCT u.user_guid FROM users u LIMIT 100) AS DistinctUUsersID LEFT JOIN (SELECT DISTINCT d.user_guid, d.dog_guid, d.breed FROM dogs d) AS DistictDUsersID ON DistinctUUsersID.user_guid=DistictDUsersID.user_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 165 rows affected.

Question 13: You might have a good guess by now about why there are duplicate rows in the dogs table and users table, even though most corporate databases are configured to prevent duplicate rows from ever being accepted. To be sure, though, let's adapt this query we wrote above:

```
SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid FROM users u) AS DistinctUUsersID
LEFT JOIN dogs d
  ON DistinctUUsersID.user_guid=d.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC
```

Add dog breed and dog weight to the columns that will be included in the final output of your query. In addition, use a HAVING clause to include only UserIDs who would have more than 10 rows in the output of the left join (your output should contain 5 rows).

In [20]:

```
%%sql
SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid FROM users u) AS DistinctUUsersID
LEFT JOIN dogs d
  ON DistinctUUsersID.user_guid=d.user_guid
GROUP BY DistinctUUsersID.user_guid
ORDER BY numrows DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[20]:

uUserID	dUserID	numrows
ce7b75bc-7144-11e5-ba71-058fbc01cf0b	ce7b75bc-7144-11e5-ba71-058fbc01cf0b	1819
ce225842-7144-11e5-ba71-058fbc01cf0b	ce225842-7144-11e5-ba71-058fbc01cf0b	26
ce2258a6-7144-11e5-ba71-058fbc01cf0b	ce2258a6-7144-11e5-ba71-058fbc01cf0b	20
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	13
ce29675e-7144-11e5-ba71-058fbc01cf0b	ce29675e-7144-11e5-ba71-058fbc01cf0b	11
ce134492-7144-11e5-ba71-058fbc01cf0b	ce134492-7144-11e5-ba71-058fbc01cf0b	9
ce6676d0-7144-11e5-ba71-058fbc01cf0b	ce6676d0-7144-11e5-ba71-058fbc01cf0b	8
ce83d2ca-7144-11e5-ba71-058fbc01cf0b	ce83d2ca-7144-11e5-ba71-058fbc01cf0b	8
ce32305a-7144-11e5-ba71-058fbc01cf0b	ce32305a-7144-11e5-ba71-058fbc01cf0b	7
ce7adeea-7144-11e5-ba71-058fbc01cf0b	ce7adeea-7144-11e5-ba71-058fbc01cf0b	7

CODE

%%sql SELECT DistinctUUsersID.user_guid AS uUserID, d.user_guid AS dUserID, count(*) AS numrows FROM (SELECT DISTINCT u.user_guid FROM users u) AS DistinctUUsersID LEFT JOIN dogs d ON DistinctUUsersID.user_guid=d.user_guid GROUP BY DistinctUUsersID.user_guid ORDER BY numrows DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 33193 rows affected.

QUESTION 13 ANSWER

In [17]:

```
%%sql
SELECT DistictUUsersID.user_guid AS userid, d.breed, d.weight, count(*) AS numrows
FROM (SELECT DISTINCT u.user_guid
FROM users u) AS DistictUUsersID
LEFT JOIN dogs d
ON DistictUUsersID.user_guid=d.user_guid
GROUP BY DistictUUsersID.user_guid
HAVING numrows>10
ORDER BY numrows DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
5 rows affected.
```

Out[17]:

userid	breed	weight	numrows
ce7b75bc-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	190	1819
ce225842-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	190	26
ce2258a6-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	190	20
ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	190	13
ce29675e-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever- Mix	60	11

You can see that almost all of the UserIDs that are causing problems are Shih Tzus that weigh 190 pounds. As we learned in earlier lessons, Dognition used this combination of breed and weight to code for testing accounts. These UserIDs do not represent real data. These types of testing entries would likely be cleaned out of databases used in large established companies, but could certainly still be present in either new databases that are still being prepared and configured, or in small companies which have not had time or resources to perfect their data storage.

There are not very many incorrect entries in the Dognition database and most of the time these entries will not appreciably affect your queries or analyses. However, you have now seen the effects such entries can have in the rare cases when you need to implement outer joins on tables that have duplicate rows or linking columns with many to many relationships. Hopefully, understanding these rare cases has helped you understand more deeply the fundamental concepts behind joining tables in relational databases.

Feel free to practice more subqueries below!

In [ ]:

MySQL Exercise 11: Queries that Test Relationships Between Test Completion and Dog Characteristics

This lesson we are going to integrate all the SQL syntax we've learned so far to start addressing questions in our Dognition Analysis Plan. I summarized the reasons having an analysis plan is so important in the "Start with an Analysis Plan" video accompanying this week's materials. Analysis plans ensure that you will address questions that are relevant to your business objectives as quickly and efficiently as possible. The quickest way to narrow in the factors in your analysis plan that are likely to create new insights is to combine simple SQL calculations with visualization programs, like Tableau, to identify which factors under consideration have the strongest effects on the business metric you are tasked with improving. You can then design more nuanced statistical models in other software, such as R, based on the factors you have confirmed are likely to be important for understanding and changing your business metric.

I describe a method for designing analysis plans in the Data Visualization and Communication with Tableau course earlier in this Specialization. I call that method Structured Pyramid Analysis Plans, or "sPAPs". I have provided a skeleton of an sPAP for the Dognition data set with the materials for this course that I will use as a road map for the queries we will design and practice in the next two lessons. To orient you, the SMART goal of the analysis project is at the top of the pyramid. This is a specific, measurable, attainable, relevant, and time-bound version of the general project objective, which is to make a recommendation to Dognition about what they could do to increase the number of tests customers complete. The variables you will use to assess the goal should be filled out right under where the SMART goal is written. Then under those variables, you will see ever-widening layers of categories and sub-categories of issues that will be important to analyze in order to achieve your SMART goal.

In this lesson, we will write queries to address the issues in the left-most branch of the sPAP. These issues all relate to "Features of Dogs" that could potentially influence the number of tests the dogs will ultimately complete. We will spend a lot of time discussing and practicing how to translate analysis questions described in words into queries written in SQL syntax.

To begin, load the sql library and database, and make the Dognition database your default database:

In [1]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[1]:

```
[]
```

In order to make it easier to practice SQL queries with meaningful examples before we learned how to join tables, I added extra columns to the "dogs" table that were not in the original Dognition database. These extra columns included the "total_tests_completed" field and multiple inter-test-interval ("iti") summary fields. **Please do NOT try to use these extra fields in the query exercises below. Since you now know how to join tables, we will practice writing queries as if you only had the data provided in the original Dognition database.**

1. Assess whether Dognition personality dimensions are related to the number of tests completed

The first variable in the Dognition sPAP we want to investigate is Dognition personality dimensions. Recall from the "Meet Your Dognition Data" video and the written description of the Dognition Data Set included with the Week 2 materials that Dognition personality dimensions represent distinct combinations of characteristics assessed by the Dognition tests. It is certainly plausible that certain personalities of dogs might be more or less likely to complete tests. For example, "einstein" dogs might be particularly likely to complete a lot of tests.

To test the relationship between Dognition personality dimensions and test completion totals, we need a query that will output a summary of the number of tests completed by dogs that have each of the Dognition personality dimensions. The features you will need to include in your query are foreshadowed by key words in this sentence. First, the fact that you need a summary of the number of tests completed suggests you will need an aggregation function. Next, the fact that you want a different summary for each personality dimension suggests that you will need a GROUP BY clause. Third, the fact that you need a "summary of the number of tests completed" rather than just a "summary of the tests completed" suggests that you might have to have multiple stages of aggegrations, which in turn might mean that you will need to use a subquery.

Let's build the query step by step.

Question 1: To get a feeling for what kind of values exist in the Dognition personality dimension column, write a query that will output all of the distinct values in the dimension column. Use your relational schema or the course materials to determine what table the dimension column is in. Your output should have 11 rows.

In [2]:

```
%%sql
SELECT DISTINCT dimension
FROM dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
11 rows affected.
```

Out[2]:

dimension
charmer
protodog
None
einstein
stargazer
maverick
socialite
ace
expert
renaissance-dog

The results of the query above illustrate there are NULL values (indicated by the output value "none") in the dimension column. Keep that in mind in case it is relevant to future queries.

We want a summary of the total number of tests completed by dogs with each personality dimension. In order to calculate those summaries, we first need to calculate the total number of tests completed by each dog. We can achieve this using a subquery. The subquery will require data from both the dogs and the complete_tests table, so the subquery will need to include a join. We are only interested in dogs who have completed tests, so an inner join is appropriate in this case.

Question 2: Use the equijoin syntax (described in MySQL Exercise 8) to write a query that will output the Dognition personality dimension and total number of tests completed by each unique DogID. This query will be used as an inner subquery in the next question. LIMIT your output to 100 rows for troubleshooting purposes.

In [6]:

```
%%sql
SELECT d.dog_guid AS dogID, d.dimension AS dimension, count(c.created_at) AS
numtests
FROM dogs d, complete_tests c
WHERE d.dog_guid=c.dog_guid
GROUP BY dogID
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[6]:

dogID	dimension	numtests
fd27b272-7144-11e5-ba71-058fbc01cf0b	charmer	21
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	protodog	20
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	None	2
fd27b79a-7144-11e5-ba71-058fbc01cf0b	None	11
fd27b86c-7144-11e5-ba71-058fbc01cf0b	einstein	31
fd27b948-7144-11e5-ba71-058fbc01cf0b	stargazer	20
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	maverick	27
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	protodog	20
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	einstein	20
fd27c5be-7144-11e5-ba71-058fbc01cf0b	socialite	20

In [8]:

```
%%sql
SELECT d.dog_guid AS dogID, d.dimension AS dimension, count(c.created_at) AS numtests
FROM dogs d, complete_tests c
WHERE d.dog_guid=c.dog_guid
GROUP BY dogID
ORDER BY numtests DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[8]:

dogID	dimension	numtests
fd793340-7144-11e5-ba71-058fbc01cf0b	charmer	55
fd7b4996-7144-11e5-ba71-058fbc01cf0b	ace	49
fd7aa428-7144-11e5-ba71-058fbc01cf0b	renaissance-dog	48
fdaefd68-7144-11e5-ba71-058fbc01cf0b	socialite	48
fd6974dc-7144-11e5-ba71-058fbc01cf0b	socialite	48
fd7bc394-7144-11e5-ba71-058fbc01cf0b	socialite	48
fd422e0e-7144-11e5-ba71-058fbc01cf0b	renaissance-dog	48
fd70d11e-7144-11e5-ba71-058fbc01cf0b	charmer	46
fd7a3628-7144-11e5-ba71-058fbc01cf0b	charmer	46
fd42a8c0-7144-11e5-ba71-058fbc01cf0b	charmer	45

CODE

%%sql SELECT d.dog_guid AS dogID, d.dimension AS dimension, count(c.created_at) AS numtests FROM dogs d, complete_tests c WHERE d.dog_guid=c.dog_guid GROUP BY dogID;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 17986 rows affected.

Question 3: Re-write the query in Question 2 using traditional join syntax (described in MySQL Exercise 8).

In [9]:

```
%%sql
SELECT d.dog_guid AS dogID, d.dimension AS dimension, count(c.created_at) AS numtests
FROM dogs d JOIN complete_tests c
  ON d.dog_guid=c.dog_guid
GROUP BY dogID
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[9]:

dogID	dimension	numtests
fd27b272-7144-11e5-ba71-058fbc01cf0b	charmer	21
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	protodog	20
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	None	2
fd27b79a-7144-11e5-ba71-058fbc01cf0b	None	11
fd27b86c-7144-11e5-ba71-058fbc01cf0b	einstein	31
fd27b948-7144-11e5-ba71-058fbc01cf0b	stargazer	20
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	maverick	27
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	protodog	20
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	einstein	20
fd27c5be-7144-11e5-ba71-058fbc01cf0b	socialite	20
fd27c74e-7144-11e5-ba71-058fbc01cf0b	None	14
fd27c7d0-7144-11e5-ba71-058fbc01cf0b	socialite	20
fd27c852-7144-11e5-ba71-058fbc01cf0b	stargazer	20
fd27c8d4-7144-11e5-ba71-058fbc01cf0b	ace	20
fd27c956-7144-11e5-ba71-058fbc01cf0b	None	11
fd27cb72-7144-11e5-ba71-058fbc01cf0b	protodog	20
fd27cd98-7144-11e5-ba71-058fbc01cf0b	expert	20
fd27ce1a-7144-11e5-ba71-058fbc01cf0b	None	7
fd27cea6-7144-11e5-ba71-058fbc01cf0b	None	2
fd27cf28-7144-11e5-ba71-058fbc01cf0b	charmer	20

Now we need to summarize the total number of tests completed by each unique DogID within each Dognition personality dimension. To do this we will need to choose an appropriate aggregation function for the count column of the query we just wrote.

Question 4: To start, write a query that will output the average number of tests completed by unique dogs in each Dognition personality dimension. Choose either the query in Question 2 or 3 to serve as an inner query in your main query. If you have trouble, make sure you use the appropriate aliases in your GROUP BY and SELECT statements.

In [10]:

```
%%sql
SELECT dimension, AVG(numtests_per_dog.numtests) AS avg_tests_completed
FROM( SELECT d.dog_guid AS dogID, d.dimension AS dimension, count(c.created_at) AS numtests
       FROM dogs d, complete_tests c
       WHERE d.dog_guid=c.dog_guid
       GROUP BY dogID) AS numtests_per_dog
GROUP BY numtests_per_dog.dimension;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
11 rows affected.
```

Out[10]:

dimension	avg_tests_completed
None	6.9416
	9.5352
ace	23.3878
charmer	23.2594
einstein	23.2171
expert	23.3926
maverick	22.8199
protodog	22.9336
renaissance-dog	23.0157
socialite	23.1194
stargazer	22.7368

OR

In [11]:

```
%%sql
SELECT dimension, AVG(numtests_per_dog.numtests) AS avg_tests_completed
FROM( SELECT d.dog_guid AS dogID, d.dimension AS dimension, count(c.created_at) AS numtests
       FROM dogs d JOIN complete_tests c
         ON d.dog_guid=c.dog_guid
       GROUP BY dogID) AS numtests_per_dog
GROUP BY numtests_per_dog.dimension;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
11 rows affected.
```

Out[11]:

dimension	avg_tests_completed
None	6.9416
	9.5352
ace	23.3878
charmer	23.2594
einstein	23.2171
expert	23.3926
maverick	22.8199
protodog	22.9336
renaissance-dog	23.0157
socialite	23.1194
stargazer	22.7368

You should retrieve an output of 11 rows with one of the dimensions labeled "None" and another labeled "" (nothing is between the quotation marks).

Question 5: How many unique DogIDs are summarized in the Dognition dimensions labeled "None" or ""? (You should retrieve values of 13,705 and 71)

In [12]:

```
%%sql
SELECT dimension, COUNT(DISTINCT dogID) AS num_dogs
FROM( SELECT d.dog_guid AS dogID, d.dimension AS dimension
      FROM dogs d JOIN complete_tests c
        ON d.dog_guid=c.dog_guid
      WHERE d.dimension IS NULL OR d.dimension=''
      GROUP BY dogID) AS dogs_in_complete_tests
GROUP BY dimension;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[12]:

dimension	num_dogs
None	13705
	71

It makes sense there would be many dogs with NULL values in the dimension column, because we learned from Dognition that personality dimensions can only be assigned after the initial "Dognition Assessment" is completed, which is comprised of the first 20 Dognition tests. If dogs did not complete the first 20 tests, they would retain a NULL value in the dimension column.

The non-NULL empty string values are more curious. It is not clear where those values would come from.

Question 6: To determine whether there are any features that are common to all dogs that have non-NULL empty strings in the dimension column, write a query that outputs the breed, weight, value in the "exclude" column, first or minimum time stamp in the complete_tests table, last or maximum time stamp in the complete_tests table, and total number of tests completed by each unique DogID that has a non-NULL empty string in the dimension column.

In [ ]:

```
CODE

%%sql
SELECT d.breed, d.weight, d.exclude, MIN(c.created_at) AS first_test, MAX(c.created_at) AS last_test,count(c.crea
ted_at) AS numtests
FROM dogs d JOIN complete_tests c
  ON d.dog_guid=c.dog_guid
WHERE d.dimension=""
GROUP BY d.dog_guid;

OUTPUT

* mysql://studentuser:***@localhost/dognitiondb
71 rows affected.
```

In [17]:

```
%%sql
SELECT d.breed, d.weight, d.exclude, MIN(c.created_at) AS first_test, MAX(c.created_at) AS last_test,count(c.crea
ted_at) AS numtests
FROM dogs d JOIN complete_tests c
  ON d.dog_guid=c.dog_guid
WHERE d.dimension=""
GROUP BY d.dog_guid
LIMIT 15;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
15 rows affected.
```

Out[17]:

breed	weight	exclude	first_test	last_test	numtests
Golden Retriever	30	0	2013-05-23 07:06:21	2013-07-02 12:15:18	17
Dachshund	10	1	2014-10-21 18:53:02	2014-10-21 19:10:07	3
Border Collie-Labrador Retriever Mix	50	0	2013-11-16 02:26:15	2013-11-16 02:38:57	4
Belgian Tervuren	70	1	2014-11-10 21:21:06	2014-12-16 01:13:28	13
Pembroke Welsh Corgi	20	1	2014-09-19 17:42:37	2014-09-22 17:58:25	4
Chihuahua	0	1	2014-10-06 00:57:46	2014-10-09 22:55:51	2
Australian Shepherd	50	1	2014-10-06 01:54:49	2014-10-30 02:16:12	14
Mixed	60	1	2014-10-10 01:01:21	2014-10-10 12:33:52	4
Portuguese Water Dog	60	1	2014-10-10 13:22:58	2014-10-10 13:36:17	3
Labrador Retriever	50	1	2014-10-06 15:28:42	2014-10-23 20:24:20	7
Golden Retriever	70	1	2014-10-06 17:27:41	2014-12-10 01:35:43	14
Poodle	50	1	2014-10-09 05:29:07	2014-11-03 03:10:19	16
Other	30	1	2014-10-21 02:05:28	2014-10-21 02:12:10	2
Labrador Retriever	70	1	2014-10-07 02:04:42	2014-10-23 02:06:02	10
West Highland White Terrier	10	1	2014-10-07 13:07:18	2014-10-11 21:57:57	19

A quick inspection of the output from the last query illustrates that almost all of the entries that have non-NULL empty strings in the dimension column also have "exclude" flags of 1, meaning that the entries are meant to be excluded due to factors monitored by the Dognition team. This provides a good argument for excluding the entire category of entries that have non-NULL empty strings in the dimension column from our analyses.

Question 7: Rewrite the query in Question 4 to exclude DogIDs with (1) non-NULL empty strings in the dimension column, (2) NULL values in the dimension column, and (3) values of "1" in the exclude column. NOTES AND HINTS: You cannot use a clause that says d.exclude does not equal 1 to remove rows that have exclude flags, because Dognition clarified that both NULL values and 0 values in the "exclude" column are valid data. A clause that says you should only include values that are not equal to 1 would remove the rows that have NULL values in the exclude column, because NULL values are never included in equals statements (as we learned in the join lessons). In addition, although it should not matter for this query, practice including parentheses with your OR and AND statements that accurately reflect the logic you intend. Your results should return 402 DogIDs in the ace dimension and 626 dogs in the charmer dimension.

In [18]:

```
%%sql
SELECT dimension, AVG(numtests_per_dog.numtests) AS avg_tests_completed, COUNT(DISTINCT dogID)
FROM( SELECT d.dog_guid AS dogID, d.dimension AS dimension, count(c.created_at) AS numtests
      FROM dogs d JOIN complete_tests c
        ON d.dog_guid=c.dog_guid
      WHERE (dimension IS NOT NULL AND dimension!='') AND (d.exclude IS NULL OR d.exclude=0)
      GROUP BY dogID) AS numtests_per_dog
GROUP BY numtests_per_dog.dimension;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
9 rows affected.
```

Out[18]:

dimension	avg_tests_completed	COUNT(DISTINCT dogID)
ace	23.5100	402
charmer	23.3594	626
einstein	23.2385	109
expert	23.4249	273
maverick	22.7673	245
protodog	22.9570	535
renaissance-dog	23.0410	463
socialite	23.0997	792
stargazer	22.7968	310

The results of Question 7 suggest there are not appreciable differences in the number of tests completed by dogs with different Dognition personality dimensions. Although these analyses are not definitive on their own, these results suggest focusing on Dognition personality dimensions will not likely lead to significant insights about how to improve Dognition completion rates.

2. Assess whether dog breeds are related to the number of tests completed

The next variable in the Dognition sPAP we want to investigate is Dog Breed. We will run one analysis with Breed Group and one analysis with Breed Type.

First, determine how many distinct breed groups there are.

Questions 8: Write a query that will output all of the distinct values in the breed_group field.

In [19]:

```
%%sql
SELECT DISTINCT breed_group
FROM dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
9 rows affected.
```

Out[19]:

breed_group
Sporting
Herding
Toy
Working
None
Hound
Non-Sporting
Terrier

You can see that there are NULL values in the breed_group field. Let's examine the properties of these entries with NULL values to determine whether they should be excluded from our analysis.

Question 9: Write a query that outputs the breed, weight, value in the "exclude" column, first or minimum time stamp in the complete_tests table, last or maximum time stamp in the complete_tests table, and total number of tests completed by each unique DogID that has a NULL value in the breed_group column.

CODE

%%sql SELECT d.breed, d.weight, d.exclude, MIN(c.created_at) AS first_test, MAX(c.created_at) AS last_test,count(c.created_at) AS numtests FROM dogs d JOIN complete_tests c ON d.dog_guid=c.dog_guid WHERE breed_group IS NULL GROUP BY d.dog_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 8816 rows affected.

In [21]:

```
%%sql
SELECT d.breed, d.weight, d.exclude, MIN(c.created_at) AS first_test, MAX(c.created_at) AS last_test,count(c.created_at) AS numtests
FROM dogs d JOIN complete_tests c
  ON d.dog_guid=c.dog_guid
WHERE breed_group IS NULL
GROUP BY d.dog_guid
LIMIT 15;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
15 rows affected.
```

Out[21]:

breed	weight	exclude	first_test	last_test	numtests
Mixed	50	None	2013-02-05 18:57:05	2013-02-05 22:38:01	20
Shih Tzu-Poodle Mix	0	None	2013-02-05 21:44:38	2013-02-10 03:33:37	20
German Shepherd Dog-Pembroke Welsh Corgi Mix	40	None	2013-02-06 04:45:28	2014-01-06 05:58:13	14
German Shepherd Dog-Nova Scotia Duck Tolling Retriever Mix	30	None	2013-05-17 17:45:46	2013-06-14 23:42:53	11
Mixed	10	None	2013-02-06 04:44:50	2013-02-06 04:48:29	2
Australian Shepherd-German Shepherd Dog Mix	90	None	2013-02-07 05:15:48	2013-12-20 21:03:18	21
Golden Doodle	70	None	2013-02-09 05:49:46	2013-02-09 06:10:11	6
Mixed	30	None	2013-02-10 03:28:12	2013-07-20 02:12:37	28
Mixed	90	1	2014-09-24 15:10:03	2014-09-24 21:23:37	20
Mudi	20	None	2014-10-06 22:21:56	2014-10-06 22:24:02	2
Parson Russell Terrier-Beagle Mix	30	None	2013-02-06 18:07:18	2013-02-06 18:16:13	4
I Don't Know	50	None	2013-02-06 22:14:00	2013-02-06 22:41:28	6
Mixed	70	None	2013-02-10 04:06:03	2015-09-28 17:33:05	45
Chihuahua- Mix	0	None	2013-02-08 04:04:51	2013-02-11 03:35:44	6
Mixed	20	None	2013-02-07 03:00:05	2013-02-07 03:16:21	4

There are a lot of these entries and there is no obvious feature that is common to all of them, so at present, we do not have a good reason to exclude them from our analysis. Therefore, let's move on to question 10 now....

Question 10: Adapt the query in Question 7 to examine the relationship between breed_group and number of tests completed. Exclude DogIDs with values of "1" in the exclude column. Your results should return 1774 DogIDs in the Herding breed group.

In [22]:

```
%%sql
SELECT breed_group, AVG(numtests_per_dog.numtests) AS avg_tests_completed, COUNT(DISTINCT dogID)
FROM( SELECT d.dog_guid AS dogID, d.breed_group AS breed_group, count(c.created_at) AS numtests
      FROM dogs d JOIN complete_tests c
        ON d.dog_guid=c.dog_guid
      WHERE d.exclude IS NULL OR d.exclude=0
      GROUP BY dogID) AS numtests_per_dog
GROUP BY breed_group;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
9 rows affected.
```

Out[22]:

breed_group	avg_tests_completed	COUNT(DISTINCT dogID)
None	10.2251	8564
	19.7542	179
Herding	11.2469	1774
Hound	10.0603	564
Non-Sporting	10.0197	964
Sporting	10.9915	2470
Terrier	9.9333	780
Toy	8.7157	1041
Working	10.2358	865

The results show there are non-NULL entries of empty strings in breed_group column again. Ignoring them for now, Herding and Sporting breed_groups complete the most tests, while Toy breed groups complete the least tests. This suggests that one avenue an analyst might want to explore further is whether it is worth it to target marketing or certain types of Dognition tests to dog owners with dogs in the Herding and Sporting breed_groups. Later in this lesson we will discuss whether using a median instead of an average to summarize the number of completed tests might affect this potential course of action.

Question 11: Adapt the query in Question 10 to only report results for Sporting, Hound, Herding, and Working breed_groups using an IN clause.

In [23]:

```
%%sql
SELECT breed_group, AVG(numtests_per_dog.numtests) AS avg_tests_completed, COUNT(DISTINCT dogID)
FROM( SELECT d.dog_guid AS dogID, d.breed_group AS breed_group, count(c.created_at) AS numtests
      FROM dogs d JOIN complete_tests c
        ON d.dog_guid=c.dog_guid
      WHERE d.exclude IS NULL OR d.exclude=0
      GROUP BY dogID) AS numtests_per_dog
GROUP BY breed_group
HAVING breed_group IN ('Sporting','Hound','Herding','Working');
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[23]:

breed_group	avg_tests_completed	COUNT(DISTINCT dogID)
Herding	11.2469	1774
Hound	10.0603	564
Sporting	10.9915	2470
Working	10.2358	865

Next, let's examine the relationship between breed_type and number of completed tests.

Questions 12: Begin by writing a query that will output all of the distinct values in the breed_type field.

In [24]:

```
%%sql
SELECT DISTINCT breed_group
FROM dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
9 rows affected.
```

Out[24]:

breed_group
Sporting
Herding
Toy
Working
None
Hound
Non-Sporting
Terrier

Question 13: Adapt the query in Question 7 to examine the relationship between breed_type and number of tests completed. Exclude DogIDs with values of "1" in the exclude column. Your results should return 8865 DogIDs in the Pure Breed group.

In [26]:

```
%%sql
SELECT breed_type, AVG(numtests_per_dog.numtests) AS avg_tests_completed, COUNT(DISTINCT dogID)
FROM( SELECT d.dog_guid AS dogID, d.breed_type AS breed_type, count(c.created_at) AS numtests
      FROM dogs d JOIN complete_tests c
        ON d.dog_guid=c.dog_guid
      WHERE d.exclude IS NULL OR d.exclude=0
      GROUP BY dogID) AS numtests_per_dog
GROUP BY breed_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[26]:

breed_type	avg_tests_completed	COUNT(DISTINCT dogID)
Cross Breed	10.6009	2884
Mixed Breed/ Other/ I Don't Know	10.2688	4818
Popular Hybrid	10.8423	634
Pure Breed	10.4107	8865

There does not appear to be an appreciable difference between number of tests completed by dogs of different breed types.

3. Assess whether dog breeds and neutering are related to the number of tests completed

To explore the results we found above a little further, let's run some queries that relabel the breed_types according to "Pure_Breed" and "Not_Pure_Breed".

Question 14: For each unique DogID, output its dog_guid, breed_type, number of completed tests, and use a CASE statement to include an extra column with a string that reads "Pure_Breed" whenever breed_type equals 'Pure Breed" and "Not_Pure_Breed" whenever breed_type equals anything else. LIMIT your output to 50 rows for troubleshooting.

CODE

%%sql SELECT d.dog_guid AS dogID, d.breed_type AS breed_type, CASE WHEN d.breed_type='Pure Breed' THEN 'pure_breed' ELSE 'not_pure_breed' END AS pure_breed,count(c.created_at) AS numtests FROM dogs d, complete_tests c WHERE d.dog_guid=c.dog_guid GROUP BY dogID;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 17986 rows affected.

In [30]:

```
%%sql
SELECT d.dog_guid AS dogID, d.breed_type AS breed_type,
     CASE WHEN d.breed_type='Pure Breed' THEN 'pure_breed'
          ELSE 'not_pure_breed'
          END AS pure_breed,count(c.created_at) AS numtests
FROM dogs d, complete_tests c
WHERE d.dog_guid=c.dog_guid
GROUP BY dogID
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[30]:

dogID	breed_type	pure_breed	numtests
fd27b272-7144-11e5-ba71-058fbc01cf0b	Pure Breed	pure_breed	21
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	Pure Breed	pure_breed	20
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	Pure Breed	pure_breed	2
fd27b79a-7144-11e5-ba71-058fbc01cf0b	Pure Breed	pure_breed	11
fd27b86c-7144-11e5-ba71-058fbc01cf0b	Pure Breed	pure_breed	31
fd27b948-7144-11e5-ba71-058fbc01cf0b	Pure Breed	pure_breed	20
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	Pure Breed	pure_breed	27
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	Mixed Breed/ Other/ I Don't Know	not_pure_breed	20
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	Pure Breed	pure_breed	20
fd27c5be-7144-11e5-ba71-058fbc01cf0b	Cross Breed	not_pure_breed	20

Question 15: Adapt your queries from Questions 7 and 14 to examine the relationship between breed_type and number of tests completed by Pure_Breed dogs and non_Pure_Breed dogs. Your results should return 8336 DogIDs in the Not_Pure_Breed group.

In [31]:

```
%%sql
SELECT numtests_per_dog.pure_breed AS pure_breed, AVG(numtests_per_dog.numtests) AS avg_tests_completed, COUNT(DI
STINCT dogID)
FROM( SELECT d.dog_guid AS dogID, d.breed_type AS breed_type,
        CASE WHEN d.breed_type='Pure Breed' THEN 'pure_breed'
             ELSE 'not_pure_breed'
             END AS pure_breed, count(c.created_at) AS numtests
      FROM dogs d JOIN complete_tests c
        ON d.dog_guid=c.dog_guid
      WHERE d.exclude IS NULL OR d.exclude=0
      GROUP BY dogID) AS numtests_per_dog
GROUP BY pure_breed;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[31]:

pure_breed	avg_tests_completed	COUNT(DISTINCT dogID)
not_pure_breed	10.4273	8336
pure_breed	10.4107	8865

Question 16: Adapt your query from Question 15 to examine the relationship between breed_type, whether or not a dog was neutered (indicated in the dog_fixed field), and number of tests completed by Pure_Breed dogs and non_Pure_Breed dogs. There are DogIDs with null values in the dog_fixed column, so your results should have 6 rows, and the average number of tests completed by non-pure-breeds who are neutered is 10.5681.

In [32]:

```
%%sql
SELECT numtests_per_dog.pure_breed AS pure_breed, neutered, AVG(numtests_per_dog.numtests) AS avg_tests_completed
, COUNT(DISTINCT dogID)
FROM( SELECT d.dog_guid AS dogID, d.breed_group AS breed_type, d.dog_fixed AS neutered,
         CASE WHEN d.breed_type='Pure Breed' THEN 'pure_breed'
         ELSE 'not_pure_breed'
         END AS pure_breed, count(c.created_at) AS numtests
      FROM dogs d JOIN complete_tests c
        ON d.dog_guid=c.dog_guid
      WHERE d.exclude IS NULL OR d.exclude=0
      GROUP BY dogID) AS numtests_per_dog
GROUP BY pure_breed, neutered;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
6 rows affected.
```

Out[32]:

pure_breed	neutered	avg_tests_completed	COUNT(DISTINCT dogID)
not_pure_breed	None	9.9897	97
not_pure_breed	0	8.6807	592
not_pure_breed	1	10.5681	7647
pure_breed	None	8.2815	135
pure_breed	0	9.3788	1687
pure_breed	1	10.6987	7043

These results suggest that although a dog's breed_type doesn't seem to have a strong relationship with how many tests a dog completed, neutered dogs, on average, seem to finish 1-2 more tests than non-neutered dogs. It may be fruitful to explore further whether this effect is consistent across different segments of dogs broken up according to other variables. If the effects are consistent, the next step would be to seek evidence that could clarify whether neutered dogs are finishing more tests due to traits that arise when a dog is neutered, or instead, whether owners who are more likely to neuter their dogs have traits that make it more likely they will want to complete more tests.

4. Other dog features that might be related to the number of tests completed, and a note about using averages as summary metrics

Two other dog features included in our sPAP were speed of game completion and previous behavioral training. Examing the relationship between the speed of game completion and number of games completed is best achieved through creating a scatter plot with a best fit line and/or running a statistical regression analysis. It is possible to achieve the statistical regression analysis through very advanced SQL queries, but the strategy that would be required is outside the scope of this course. Therefore, I would recommend exporting relevant data to a program like Tableau, R, or Matlab in order to assess the relationship between the speed of game completion and number of games completed.

Unfortunately, there is no field available in the Dognition data that is relevant to a dog's previous behavioral training, so more data would need to be collected to examine whether previous behavioral training is related to the number of Dognition tests completed.

One last issue I would like to address in this lesson is the issue of whether an average is a good summary to use to represent the values of a certain group. Average calculations are very sensitive to extreme values, or outliers, in the data. This video provides a nice demonstration of how sensitive averages can be:

http://www.statisticslectures.com/topics/outliereffects/ (http://www.statisticslectures.com/topics/outliereffects/)

Ideally, you would summarize the data in a group using a median calculation when you either don't know the distribution of values in your data or you already know that outliers are present (the definition of median is covered in the video above). Unfortunately, medians are more computationally intensive than averages, and there is no pre-made function that allows you to calculate medians using SQL. If you wanted to calculate the median, you would need to use an advanced strategy such as the ones described here:

https://www.periscopedata.com/blog/medians-in-sql.html (https://www.periscopedata.com/blog/medians-in-sql.html)

Despite the fact there is no simple way to calculate medians using SQL, there is a way to get a hint about whether average values are likely to be wildly misleading. As described in the first video (http://www.statisticslectures.com/topics/outliereffects/ (http://www.statisticslectures.com/topics/outliereffects/)), strong outliers lead to large standard deviation values. Fortunately, we *CAN* calculate standard deviations in SQL easily using the STDDEV function. Therefore, it is good practice to include standard deviation columns with your outputs so that you have an idea whether the average values outputted by your queries are trustworthy. Whenever standard deviations are a significant portion of the average values of a field, and certainly when standard deviations are larger than the average values of a field, it's a good idea to export your data to a program that can handle more sophisticated statistical analyses before you interpret any results too strongly.

Let's practice including standard deviations in our queries and interpretting their values.

Question 17: Adapt your query from Question 7 to include a column with the standard deviation for the number of tests completed by each Dognition personality dimension.

In [ ]:

```
%%sql
SELECT dimension, AVG(numtests) AS avg_tests_completed, COUNT(DISTINCT dogID), STDDEV(numtests)
FROM( SELECT d.dog_guid AS dogID, d.dimension AS dimension, count(c.created_at) AS numtests
      FROM dogs d JOIN complete_tests c
        ON d.dog_guid=c.dog_guid
      WHERE (dimension IS NOT NULL AND dimension!='') AND (d.exclude IS NULL OR d.exclude=0)
      GROUP BY dogID) AS numtests_per_dog
GROUP BY numtests_per_dog.dimension;
```

The standard deviations are all around 20-25% of the average values of each personality dimension, and they are not appreciably different across the personality dimensions, so the average values are likely fairly trustworthy. Let's try calculating the standard deviation of a different measurement.

Question 18: Write a query that calculates the average amount of time it took each dog breed_type to complete all of the tests in the exam_answers table. Exclude negative durations from the calculation, and include a column that calculates the standard deviation of durations for each breed_type group:

In [33]:

```
%%sql
SELECT d.breed_type AS breed_type, AVG(TIMESTAMPDIFF(minute,e.start_time,e.end_time)) AS AvgDuration,
       STDDEV(TIMESTAMPDIFF(minute,e.start_time,e.end_time)) AS StdDevDuration
FROM dogs d JOIN exam_answers e
  ON d.dog_guid=e.dog_guid
WHERE TIMESTAMPDIFF(minute,e.start_time,e.end_time)>0
GROUP BY breed_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[33]:

breed_type	AvgDuration	StdDevDuration
Cross Breed	11810.3230	59113.45580229881
Mixed Breed/ Other/ I Don't Know	9145.1575	48748.626840777506
Popular Hybrid	7734.0763	45577.65824281632
Pure Breed	12311.2558	60997.35425304078

This time many of the standard deviations have larger magnitudes than the average duration values. This suggests there are outliers in the data that are significantly impacting the reported average values, so the average values are not likely trustworthy. These data should be exported to another program for more sophisticated statistical analysis.

In the next lesson, we will write queries that assess the relationship between testing circumstances and the number of tests completed. Until then, feel free to practice any additional queries you would like to below!

In [ ]:

```

```

MySQL Exercise 2: Using WHERE to select specific data

When you are querying a business-related data set, you are usually doing so to answer a question about a subset of the data. In this lesson you will learn how to select subsets of rows of data that meet criteria you specify, to help you prepare for these types of business questions. The mechanism within a SQL query that allows you specify which subset of data you want to retrieve is the WHERE clause.

Before we begin, let's load the SQL library and Dognition database, and make the Dognition database our default database. As a reminder, these are the lines of code you should input:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

In [2]:

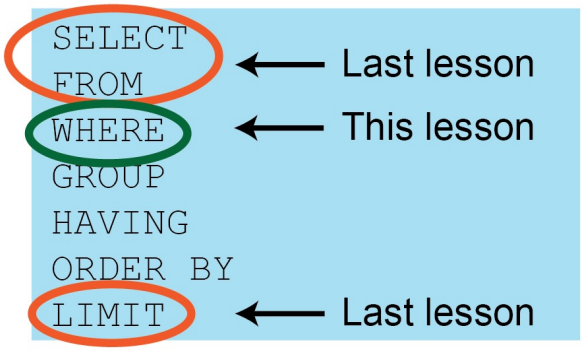
```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[2]:

```
[]
```

Recall the general syntax structure we learned from the "Introduction to Query Syntax" video at the beginning of the week:



This guide indicates that whenever the data we select need to meet certain criteria (specified using a "WHERE" clause), we specify those criteria after we have specified where the data come from.

Let's say we want to know which Dognition customers received access to Dognition's first four tests for free. These customers have a 1 in the "free_start_user" column of the users table. The syntax you would use to select the data for these customers would be:

```
SELECT user_guid
FROM users
WHERE free_start_user=1;
```

(Note: user_guid is the field that specifies the unique User ID number of each customer in the users table)

If you wanted to double-check that the outputted data indeed met the criteria you specified, you could include a second column in your output that would give you the value in the free_start_user field for each row of the output:

```
SELECT user_guid, free_start_user
FROM users
WHERE free_start_user=1;
```

Try this on your own below. Remember to use %%sql to indicate that your query will span multiple lines, and consider whether you would like to limit the number of results you ouput using the syntax we learned last lesson. If you do use a LIMIT statement, remember that it has to be the last item in your query, so this time you will place it after your WHERE statement instead of after your FROM statement.

In [17]:

```
%%sql
SELECT user_guid
FROM users
WHERE free_start_user=1 LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[17]:

user_guid
ce28a468-7144-11e5-ba71-058fbc01cf0b
ce28ac4c-7144-11e5-ba71-058fbc01cf0b
ce28acba-7144-11e5-ba71-058fbc01cf0b
ce28ad1e-7144-11e5-ba71-058fbc01cf0b
ce28ad82-7144-11e5-ba71-058fbc01cf0b
ce28b098-7144-11e5-ba71-058fbc01cf0b
ce28b1c4-7144-11e5-ba71-058fbc01cf0b
ce28b58e-7144-11e5-ba71-058fbc01cf0b
ce28b9bc-7144-11e5-ba71-058fbc01cf0b
ce28ba20-7144-11e5-ba71-058fbc01cf0b

In [16]:

```
%%sql
SELECT user_guid, free_start_user
FROM users
WHERE free_start_user=1 LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[16]:

user_guid	free_start_user
ce28a468-7144-11e5-ba71-058fbc01cf0b	1
ce28ac4c-7144-11e5-ba71-058fbc01cf0b	1
ce28acba-7144-11e5-ba71-058fbc01cf0b	1
ce28ad1e-7144-11e5-ba71-058fbc01cf0b	1
ce28ad82-7144-11e5-ba71-058fbc01cf0b	1
ce28b098-7144-11e5-ba71-058fbc01cf0b	1
ce28b1c4-7144-11e5-ba71-058fbc01cf0b	1
ce28b58e-7144-11e5-ba71-058fbc01cf0b	1
ce28b9bc-7144-11e5-ba71-058fbc01cf0b	1
ce28ba20-7144-11e5-ba71-058fbc01cf0b	1

Question 1: How would you select the Dog IDs for the dogs in the Dognition data set that were DNA tested (these should have a 1 in the dna_tested field of the dogs table)? Try it below (if you do not limit your output, your query should output data from 1433 dogs):

In [15]:

```
%%sql
SELECT dog_guid
FROM dogs
WHERE dna_tested=1 LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[15]:

dog_guid
fd27b6b4-7144-11e5-ba71-058fbc01cf0b
fd27cd98-7144-11e5-ba71-058fbc01cf0b
fd27ce1a-7144-11e5-ba71-058fbc01cf0b
fd27d144-7144-11e5-ba71-058fbc01cf0b
fd27d1c6-7144-11e5-ba71-058fbc01cf0b
fd27d9fa-7144-11e5-ba71-058fbc01cf0b
fd27dc52-7144-11e5-ba71-058fbc01cf0b
fd27e454-7144-11e5-ba71-058fbc01cf0b
fd27e9a4-7144-11e5-ba71-058fbc01cf0b
fd3cd40e-7144-11e5-ba71-058fbc01cf0b

The SELECT statement can be used to interact with all data types, and there are many operators and functions that allow you to interact with the data in different ways. Here are some resources that describe these operators and functions:

https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html (https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html)
http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

Some of the most common operators include: =,<,>,<=, and >=. If you want to select something that is NOT a specific value, use != or <>. You can also use logical operators, such as AND and OR.

Let's start by examining how operators can be used with numerical data.

If you wanted to examine the Dog IDs of dogs who weighed between 10 and 50 pounds, you could query:

```
SELECT dog_guid, weight
FROM dogs
WHERE weight BETWEEN 10 AND 50;
```

The above query provided an example of how to use the BETWEEN operator (described in the links provided above), as well as an example of how AND can be used to specify multiple criteria. If you wanted to examine the Dog IDs of dogs who were "fixed" (neutered) OR DNA tested, you could use OR in the following query:

```
SELECT dog_guid, dog_fixed, dna_tested
FROM dogs
WHERE dog_fixed=1 OR dna_tested=1;
```

If you wanted to examine the Dog IDs of dogs who were fixed but NOT DNA tested, you could query:

```
SELECT dog_guid, dog_fixed, dna_tested
FROM dogs
WHERE dog_fixed=1 AND dna_tested!=1;
```

Question 2: How would you query the User IDs of customers who bought annual subscriptions, indicated by a "2" in the membership_type field of the users table? (If you do not limit the output of this query, your output should contain 4919 rows.)

In [14]:

```
%%sql
SELECT user_guid
FROM users
WHERE membership_type=2 LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[14]:

user_guid
ce134e42-7144-11e5-ba71-058fbc01cf0b
ce135e14-7144-11e5-ba71-058fbc01cf0b
ce135e14-7144-11e5-ba71-058fbc01cf0b
ce136ac6-7144-11e5-ba71-058fbc01cf0b
ce136c24-7144-11e5-ba71-058fbc01cf0b
ce136e36-7144-11e5-ba71-058fbc01cf0b
ce136ee0-7144-11e5-ba71-058fbc01cf0b
ce136f94-7144-11e5-ba71-058fbc01cf0b
ce134be0-7144-11e5-ba71-058fbc01cf0b
ce1371a6-7144-11e5-ba71-058fbc01cf0b

Now let's try using the WHERE statement to interact with text data (called "strings").

Strings need to be surrounded by quotation marks in SQL. MySQL accepts both double and single quotation marks, but some database systems only accept single quotation marks. Whenever a string contains an SQL keyword, the string must be enclosed in backticks instead of quotation marks.

```
'the marks that surrounds this phrase are single quotation marks'
"the marks that surrounds this phrase are double quotation marks"
`the marks that surround this phrase are backticks`
```

Strings enclosed in quotation or backticks can be used with many of the same operators as numerical data. For example, imagine that you only wanted to look at data from dogs of the breed "Golden Retrievers." You could query (note that double quotation marks could have been used in this example is well):

```
SELECT dog_guid, breed
FROM dogs
WHERE breed='golden retriever';
```

The IN operator allows you to specify multiple values in a WHERE clause. Each of these values must be separated by a comma from the other values, and the entire list of values should be enclosed in parentheses. If you wanted to look at all the data from Golden Retrievers and Poodles, you could certainly use the OR operator, but the IN operator would be even more efficient (note that single quotation marks could have been used in this example, too):

```
SELECT dog_guid, breed
FROM dogs
WHERE breed IN ("golden retriever","poodle");
```

The LIKE operator allows you to specify a pattern that the textual data you query has to match. For example, if you wanted to look at all the data from breeds whose names started with "s", you could query:

```
SELECT dog_guid, breed
FROM dogs
WHERE breed LIKE ("s%");
```

In this syntax, the percent sign indicates a wild card. Wild cards represent unlimited numbers of missing letters. This is how the placement of the percent sign would affect the results of the query:

- WHERE breed LIKE ("s%") = the breed must start with "s", but can have any number of letters after the "s"
- WHERE breed LIKE ("%s") = the breed must end with "s", but can have any number of letters before the "s"
- WHERE breed LIKE ("%s%") = the breed must contain an "s" somewhere in its name, but can have any number of letters before or after the "s"

Question 3: How would you query all the data from customers located in the state of North Carolina (abbreviated "NC") or New York (abbreviated "NY")? If you do not limit the output of this query, your output should contain 1333 rows.

In [13]:

```
%%sql
SELECT *
FROM users
WHERE state IN ("NC","NY") LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[13]:

sign_in_count	created_at	updated_at	max_dogs	membership_id	subscribed	exclude	free_start_user	last_active_at	membership_type
181	2013-02-05 17:54:42	2015-01-28 20:51:49	13	2	1	1	0	None	2
181	2013-02-05 17:54:42	2015-01-28 20:51:49	13	2	1	1	0	None	2
65	2013-02-05 00:52:16	2015-01-28 20:51:49	3	2	1	None	None	None	2
7	2013-02-06 00:40:59	2015-01-28 20:51:50	1	2	1	None	None	None	2
15	2013-02-06 14:13:42	2015-01-28 20:51:50	1	2	1	None	None	None	2
181	2013-02-05 17:54:42	2015-01-28 20:51:49	13	2	1	1	0	None	2
2	2013-02-06 19:50:16	2015-01-28 20:51:50	2	2	1	None	None	None	2
181	2013-02-05 17:54:42	2015-01-28 20:51:49	13	2	1	1	0	None	2
4	2013-02-07 04:19:57	2015-01-28 20:51:50	2	1	0	None	None	2014-06-23 20:42:36	1
2	2013-02-06 01:54:46	2015-01-28 20:51:50	1	1	0	None	None	None	1

Next, let's try using the WHERE statement to interact with datetime data. Time-related data is a little more complicated to work with than other types of data, because it must have a very specific format. MySQL comes with the following data types for storing a date or a date/time value in the database:

DATE - format YYYY-MM-DD
DATETIME - format: YYYY-MM-DD HH:MI:SS
TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
YEAR - format YYYY or YY

One of the interesting things about time-related data is that SQL has commands to break the data into different "time parts" or "date parts" as described here:

http://www.tutorialspoint.com/mysql/mysql-date-time-functions.htm (http://www.tutorialspoint.com/mysql/mysql-date-time-functions.htm)

A time stamp stored in one row of data might look like this:

```
2013-02-07 02:50:52
```

The year part of that entry would be 2013, the month part would be "02" or "February" (depending on the requested format), the seconds part would be "52", and so on. SQL functions easily allow you to convert those parts into formats you might need for specific analyses. For example, imagine you wanted to know how many tests Dognition customers complete on different days of the week. To complete this analysis, you would need to convert the time stamps of each completed test to a variable that outputted the correct day of the week for that date. DAYNAME is a function that will do this for you. You can combine DAYNAME with WHERE to select data from only a single day of the week:

```
SELECT dog_guid, created_at
FROM complete_tests
WHERE DAYNAME(created_at)="Tuesday"
```

You can also use common operators like =,<,>,<=,>=,!=, or <> with dates just like you would with other types of data, but whether you refer to the date as a number or text will depend on whether you are selecting individual date parts or treating the date/time entry as a single clause. For example, you could select all the Dog IDs and time stamps of tests completed after the 15 of every month with this command that extracts the "DAY" date part out of each time stamp:

```
SELECT dog_guid, created_at
FROM complete_tests
WHERE DAY(created_at) > 15
```

You could also select all the Dog IDs and time stamps of completed tests from after February 4, 2014 by treating date entries as text clauses with the following query:

```
SELECT dog_guid, created_at
FROM complete_tests
WHERE created_at > '2014-02-04'
```

Note that you have to use a different set of functions than you would use for regular numerical data to add or subtract time from any values in these datetime formats. For example, instead of using a minus sign to find the difference in time between two time stamps or dates, you would use the TIMEDIFF or DATEDIFF function. See the references provided above for a list of these functions.

Question 4: Now that you have seen how datetime data can be used to impose criteria on the data you select, how would you select all the Dog IDs and time stamps of Dognition tests completed before October 15, 2015 (your output should have 193,246 rows)?

In [18]:

```
%%sql
SELECT dog_guid, created_at
FROM complete_tests
WHERE created_at < '2015-10-15' LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[18]:

dog_guid	created_at
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:26:54
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:31:03
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:32:04
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:32:25
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:32:56
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:33:15
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:33:33
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:33:59
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:34:25
fd27b86c-7144-11e5-ba71-058fbc01cf0b	2013-02-05 18:34:39

Last, let's use the WHERE statement in combination with two very important operators: IS NULL and IS NOT NULL. IS NULL will indicate rows of data that have null values. IS NOT NULL will indicate rows that do not have null values. We saw in previous exercises that many of the entries in the free_start_user field of the user table in the Dognition data set had NULL values. To select only the rows that have non-null data you could query:

```
SELECT user_guid
FROM users
WHERE free_start_user IS NOT NULL;
```

To select only the rows that have null data so that you can examine if these rows share something else in common, you could query:

```
SELECT user_guid
FROM users
WHERE free_start_user IS NULL;
```

Question 5: How would you select all the User IDs of customers who do not have null values in the State field of their demographic information (if you do not limit the output, you should get 17,985 from this query -- there are a lot of null values in the state field!)?

In [20]:

```
%%sql
SELECT user_guid
FROM users
WHERE state IS NOT NULL LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[20]:

user_guid
ce134e42-7144-11e5-ba71-058fbc01cf0b
ce1353d8-7144-11e5-ba71-058fbc01cf0b
ce135ab8-7144-11e5-ba71-058fbc01cf0b
ce13507c-7144-11e5-ba71-058fbc01cf0b
ce135e14-7144-11e5-ba71-058fbc01cf0b
ce13615c-7144-11e5-ba71-058fbc01cf0b
ce135e14-7144-11e5-ba71-058fbc01cf0b
ce135f2c-7144-11e5-ba71-058fbc01cf0b
ce136a1c-7144-11e5-ba71-058fbc01cf0b
ce136ac6-7144-11e5-ba71-058fbc01cf0b

Practice writing your own SELECT and WHERE statements!

These queries will combine what you've learned in the past two lessons.

Question 6: How would you retrieve the Dog ID, subcategory_name, and test_name fields, in that order, of the first 10 reviews entered in the Reviews table to be submitted in 2014?

In [22]:

```
%%sql
SELECT dog_guid, subcategory_name, test_name
FROM reviews
WHERE YEAR(created_at)='2014'
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[22]:

dog_guid	subcategory_name	test_name
ce3ac77e-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Warm-up
ce2aedcc-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Warm-up
ce2aedcc-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Game
ce2aedcc-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm-up
ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Warm-up
ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Yawn Game
ce405c52-7144-11e5-ba71-058fbc01cf0b	Empathy	Eye Contact Game
ce405e28-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm-up
ce405e28-7144-11e5-ba71-058fbc01cf0b	Cunning	Turn Your Back
ce2609c4-7144-11e5-ba71-058fbc01cf0b	Communication	Treat Warm-up

Question 7: How would you select all of the User IDs of customers who have female dogs whose breed includes the word "terrier" somewhere in its name (if you don't limit your output, you should have 1771 rows in your output)?

In [5]:

```
%%sql
SELECT dog_guid, gender, breed
FROM dogs
WHERE gender='female' and breed LIKE ("%terrier%")
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[5]:

dog_guid	gender	breed
fd27eae4-7144-11e5-ba71-058fbc01cf0b	female	Australian Terrier
fd3cf8ee-7144-11e5-ba71-058fbc01cf0b	female	Bedlington Terrier
fd3cfa1a-7144-11e5-ba71-058fbc01cf0b	female	Russell Terrier
fd3d080c-7144-11e5-ba71-058fbc01cf0b	female	Boston Terrier-Chihuahua Mix
fd3d0898-7144-11e5-ba71-058fbc01cf0b	female	American Pit Bull Terrier
fd3d2080-7144-11e5-ba71-058fbc01cf0b	female	Rat Terrier
fd3d2116-7144-11e5-ba71-058fbc01cf0b	female	Border Terrier
fd3d2af8-7144-11e5-ba71-058fbc01cf0b	female	Maltese-Yorkshire Terrier Mix
fd3d34d0-7144-11e5-ba71-058fbc01cf0b	female	Parson Russell Terrier
fd3d3840-7144-11e5-ba71-058fbc01cf0b	female	West Highland White Terrier

Question 8: How would you select the Dog ID, test name, and subcategory associated with each completed test for the first 100 tests entered in October, 2014?

In [4]:

```
%%sql
SELECT dog_guid, test_name, Subcategory_name
FROM complete_tests
WHERE YEAR(created_at)='2014' and MONTH(created_at)='10'
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[4]:

dog_guid	test_name	Subcategory_name
fd6a3480-7144-11e5-ba71-058fbc01cf0b	Delayed Cup Game	Memory
fd6a3480-7144-11e5-ba71-058fbc01cf0b	Inferential Reasoning Warm-up	Reasoning
fd6a3480-7144-11e5-ba71-058fbc01cf0b	Inferential Reasoning Game	Reasoning
fd6a3480-7144-11e5-ba71-058fbc01cf0b	Physical Reasoning Warm-up	Reasoning
fd6a3480-7144-11e5-ba71-058fbc01cf0b	Physical Reasoning Game	Reasoning
fd6a2350-7144-11e5-ba71-058fbc01cf0b	Memory versus Smell	Memory
fd6924aa-7144-11e5-ba71-058fbc01cf0b	Yawn Warm-up	Empathy
fd6924aa-7144-11e5-ba71-058fbc01cf0b	Yawn Game	Empathy
fd6924aa-7144-11e5-ba71-058fbc01cf0b	Eye Contact Warm-up	Empathy
fd6924aa-7144-11e5-ba71-058fbc01cf0b	Eye Contact Game	Empathy

There are many more operators you can use in your WHERE clauses to restrict the data you select as well. We do not have the space to go over each one individually in this lesson, but I encourage you to explore them on your own. *This is a great area to practice being fearless and bold in your desire to learn new things! The more you try, the more you will learn.*

Feel free to practice any other functions or operators you discover in the space below:

In [ ]:

```

```

MySQL Exercise 3: Formatting Selected Data

In this lesson, we are going to learn about three SQL clauses or functionalities that will help you format and edit the output of your queries. We will also learn how to export the results of your formatted queries to a text file so that you can analyze them in other software packages such as Tableau or Excel.

Begin by loading the SQL library into Jupyter, connecting to the Dognition database, and setting Dognition as the default database.

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

In [2]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[2]:

```
[]
```

1. Use AS to change the titles of the columns in your output

The AS clause allows you to assign an alias (a temporary name) to a table or a column in a table. Aliases can be useful for increasing the readability of queries, for abbreviating long names, and for changing column titles in query outputs. To implement the AS clause, include it in your query code immediately after the column or table you want to rename. For example, if you wanted to change the name of the time stamp field of the completed_tests table from "created_at" to "time_stamp" in your output, you could take advantage of the AS clause and execute the following query:

```
SELECT dog_guid, created_at AS time_stamp
FROM complete_tests
```

Note that if you use an alias that includes a space, the alias must be surrounded in quotes:

```
SELECT dog_guid, created_at AS "time stamp"
FROM complete_tests
```

You could also make an alias for a table:

```
SELECT dog_guid, created_at AS "time stamp"
FROM complete_tests AS tests
```

Since aliases are strings, again, MySQL accepts both double and single quotation marks, but some database systems only accept single quotation marks. It is good practice to avoid using SQL keywords in your aliases, but if you have to use an SQL keyword in your alias for some reason, the string must be enclosed in backticks instead of quotation marks.

Question 1: How would you change the title of the "start_time" field in the exam_answers table to "exam start time" in a query output? Try it below:

In [3]:

```
%%sql
SELECT start_time AS exam_start_time
FROM exam_answers
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2460320 rows affected.
```

2. Use DISTINCT to remove duplicate rows

Especially in databases like the Dognition database where no primary keys were declared in each table, sometimes entire duplicate rows can be entered in error. Even with no duplicate rows present, sometimes your queries correctly output multiple instances of the same value in a column, but you are interested in knowing what the different possible values in the column are, not what each value in each row is. In both of these cases, the best way to arrive at the clean results you want is to instruct the query to return only values that are distinct, or different from all the rest. The SQL keyword that allows you to do this is called DISTINCT. To use it in a query, place it directly after the word SELECT in your query.

For example, if we wanted a list of all the breeds of dogs in the Dognition database, we could try the following query from a previous exercise:

```
SELECT breed
FROM dogs;
```

However, the output of this query would not be very helpful, because it would output the entry for every single row in the breed column of the dogs table, regardless of whether it duplicated the breed of a previous entry. Fortunately, we could arrive at the list we want by executing the following query with the DISTINCT modifier:

```
SELECT DISTINCT breed
FROM dogs;
```

Try it yourself (If you do not limit your output, you should get 2006 rows in your output):

In [4]:

```
%%sql
SELECT DISTINCT breed
FROM dogs
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[4]:

breed
Labrador Retriever
Shetland Sheepdog
Golden Retriever
Shih Tzu
Siberian Husky
Mixed
Shih Tzu-Poodle Mix
German Shepherd Dog-Pembroke Welsh Corgi Mix
Vizsla
Pug

If you scroll through the output, you will see that no two entries are the same. Of note, if you use the DISTINCT clause on a column that has NULL values, MySQL will include one NULL value in the DISTINCT output from that column.

When the DISTINCT clause is used with multiple columns in a SELECT statement, the combination of all the columns together is used to determine the uniqueness of a row in a result set.

For example, if you wanted to know all the possible combinations of states and cities in the users table, you could query:

```
SELECT DISTINCT state, city
FROM users;
```

Try it (if you don't limit your output you'll see 3999 rows in the query result, of which the first 1000 are displayed):

In [5]:

```
%%sql
SELECT DISTINCT State, city
FROM users
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[5]:

State	city
ND	Grand Forks
MA	Barre
CT	Darien
IL	Winnetka
NC	Raleigh
WA	Auburn
CO	Fort Collins
WA	Seattle
WA	Bainbridge Island
WA	Bremerton

If you examine the query output carefully, you will see that there are many rows with California (CA) in the state column and four rows that have Gainesville in the city column (Georgia, Arkansas, Florida, and Virginia all have cities named Gainesville in our user table), but no two rows have the same state and city combination.

When you use the DISTINCT clause with the LIMIT clause in a statement, MySQL stops searching when it finds the number of *unique* rows specified in the LIMIT clause, not when it goes through the number of rows in the LIMIT clause.

For example, if the first 6 entries of the breed column in the dogs table were:

Labrador Retriever
Shetland Sheepdog
Golden Retriever
Golden Retriever
Shih Tzu
Siberian Husky

The output of the following query:

```
SELECT DISTINCT breed
FROM dogs LIMIT 5;
```

would be the first 5 different breeds:

Labrador Retriever
Shetland Sheepdog
Golden Retriever
Shih Tzu
Siberian Husky

not the distinct breeds in the first 5 rows:

Labrador Retriever
Shetland Sheepdog
Golden Retriever
Shih Tzu

Question 2: How would you list all the possible combinations of test names and subcategory names in complete_tests table? (If you do not limit your output, you should retrieve 45 possible combinations)

In [24]:

```
%%sql
SELECT DISTINCT test_name, subcategory_name
FROM complete_tests
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

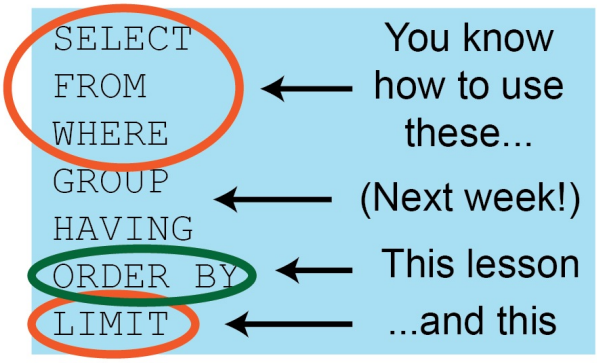
Out[24]:

test_name	subcategory_name
Yawn Warm-up	Empathy
Yawn Game	Empathy
Eye Contact Warm-up	Empathy
Eye Contact Game	Empathy
Treat Warm-up	Communication
Arm Pointing	Communication
Foot Pointing	Communication
Watching	Cunning
Turn Your Back	Cunning
Cover Your Eyes	Cunning
Watching - Part 2	Cunning
One Cup Warm-up	Memory
Two Cup Warm-up	Memory
Memory versus Pointing	Memory
Memory versus Smell	Memory
Delayed Cup Game	Memory
Inferential Reasoning Warm-up	Reasoning
Inferential Reasoning Game	Reasoning
Physical Reasoning Warm-up	Reasoning
Physical Reasoning Game	Reasoning

3. Use ORDER BY to sort the output of your query

As you might have noticed already when examining the output of the queries you have executed thus far, databases do not have built-in sorting mechanisms that automatically sort the output of your query. However, SQL permits the use of the powerful ORDER BY clause to allow you to sort the output according to your own specifications. Let's look at how you would implement a simple ORDER BY clause.

Recall our query outline:



Your ORDER BY clause will come after everything else in the main part of your query, but before a LIMIT clause.

If you wanted the breeds of dogs in the dog table sorted in alphabetical order, you could query:

```
SELECT DISTINCT breed
FROM dogs
ORDER BY breed
```

Try it yourself:

In [25]:

```
%%sql
SELECT DISTINCT breed
FROM dogs
ORDER BY breed
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[25]:

breed
-American Eskimo Dog Mix
-American Pit Bull Terrier Mix
-Anatolian Shepherd Dog Mix
-Australian Cattle Dog Mix
-Australian Shepherd Mix
-Beagle Mix
-Bichon Frise Mix
-Bluetick Coonhound Mix
-Border Collie Mix
-Boxer Mix
-Cairn Terrier Mix
-Cavalier King Charles Spaniel Mix
-Chesapeake Bay Retriever Mix
-Chihuahua Mix
-Cocker Spaniel Mix
-Collie Mix
-German Shepherd Dog Mix
-Golden Retriever Mix
-Great Dane Mix
-Great Pyrenees Mix

(You might notice that some of the breeds start with a hyphen; we'll come back to that later.)

The default is to sort the output in ascending order. However, you can tell SQL to sort the output in descending order as well:

```
SELECT DISTINCT breed
FROM dogs
ORDER BY breed DESC
```

Combining ORDER BY with LIMIT gives you an easy way to select the "top 10" and "last 10" in a list or column. For example, you could select the User IDs and Dog IDs of the 5 customer-dog pairs who spent the least median amount of time between their Dognition tests:

```
SELECT DISTINCT user_guid, median_ITI_minutes
FROM dogs
ORDER BY median_ITI_minutes
LIMIT 5
```

or the greatest median amount of time between their Dognition tests:

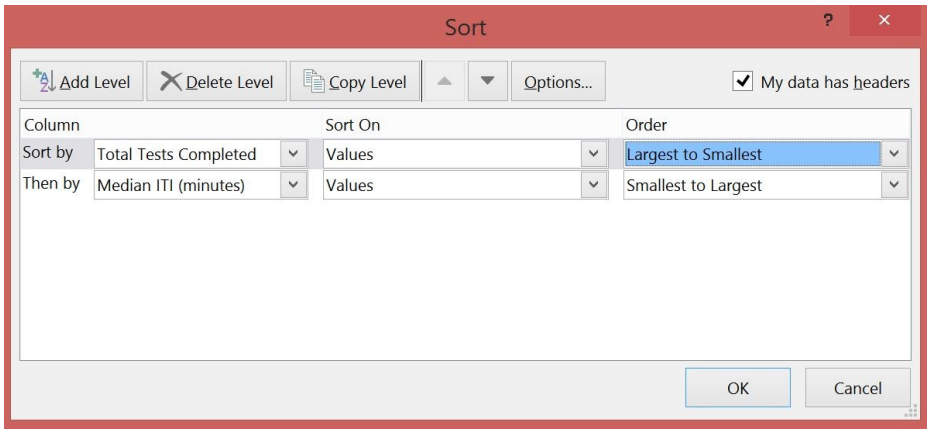
```
SELECT DISTINCT user_guid, median_ITI_minutes
FROM dogs
ORDER BY median_ITI_minutes DESC
LIMIT 5
```

You can also sort your output based on a derived field. If you wanted your inter-test interval to be expressed in seconds instead of minutes, you could incorporate a derived column and an alias into your last query to get the 5 customer-dog pairs who spent the greatest median amount of time between their Dognition tests in seconds:

```
SELECT DISTINCT user_guid, (median_ITI_minutes * 60) AS median_ITI_sec
FROM dogs
ORDER BY median_ITI_sec DESC
LIMIT 5
```

Note that the parentheses are important in that query; without them, the database would try to make an alias for 60 instead of median_ITI_minutes * 60.

SQL queries also allow you to sort by multiple fields in a specified order, similar to how Excel allows to include multiple levels in a sort (see image below):



To achieve this in SQL, you include all the fields (or aliases) by which you want to sort the results after the ORDER BY clause, separated by commas, in the order you want them to be used for sorting. You can then specify after each field whether you want the sort using that field to be ascending or descending.

If you wanted to select all the distinct User IDs of customers in the United States (abbreviated "US") and sort them according to the states they live in in alphabetical order first, and membership type second, you could query:

```
SELECT DISTINCT user_guid, state, membership_type
FROM users
WHERE country="US"
ORDER BY state ASC, membership_type ASC
```

Go ahead and try it yourself (if you do not limit the output, you should get 9356 rows in your output):

In [8]:

```
%%sql
SELECT DISTINCT user_guid, state, membership_type
FROM users
WHERE country="US"
ORDER BY state ASC, membership_type ASC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[8]:

user_guid	state	membership_type
ce138312-7144-11e5-ba71-058fbc01cf0b	AE	1
ce7587ba-7144-11e5-ba71-058fbc01cf0b	AE	1
ce76f528-7144-11e5-ba71-058fbc01cf0b	AE	1
ce221dbe-7144-11e5-ba71-058fbc01cf0b	AE	2
ce70836e-7144-11e5-ba71-058fbc01cf0b	AE	2
ce969298-7144-11e5-ba71-058fbc01cf0b	AE	3
ce26e01a-7144-11e5-ba71-058fbc01cf0b	AK	1
ce351572-7144-11e5-ba71-058fbc01cf0b	AK	1
ce3c3b36-7144-11e5-ba71-058fbc01cf0b	AK	1
ce4170ec-7144-11e5-ba71-058fbc01cf0b	AK	1
ce666e38-7144-11e5-ba71-058fbc01cf0b	AK	1
ce7007a4-7144-11e5-ba71-058fbc01cf0b	AK	1
ce718782-7144-11e5-ba71-058fbc01cf0b	AK	1
ce72c64c-7144-11e5-ba71-058fbc01cf0b	AK	1
ce72ef5a-7144-11e5-ba71-058fbc01cf0b	AK	1
ce735210-7144-11e5-ba71-058fbc01cf0b	AK	1
ce741718-7144-11e5-ba71-058fbc01cf0b	AK	1
ce74f656-7144-11e5-ba71-058fbc01cf0b	AK	1
ce756d52-7144-11e5-ba71-058fbc01cf0b	AK	1
ce77149a-7144-11e5-ba71-058fbc01cf0b	AK	1

You might notice that some of the rows have null values in the state field. You could revise your query to only select rows that do not have null values in either the state or membership_type column:

```
SELECT DISTINCT user_guid, state, membership_type
FROM users
WHERE country="US" AND state IS NOT NULL and membership_type IS NOT NULL
ORDER BY state ASC, membership_type ASC
```

Question 3: Below, try executing a query that would sort the same output as described above by membership_type first in descending order, and state second in ascending order:

In [11]:

```
%%sql
SELECT DISTINCT user_guid, state, membership_type
FROM users
WHERE country ="US" and state is not NULL and membership_type is not NULL
ORDER BY membership_type ASC, state ASC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[11]:

user_guid	state	membership_type
ce138312-7144-11e5-ba71-058fbc01cf0b	AE	1
ce7587ba-7144-11e5-ba71-058fbc01cf0b	AE	1
ce76f528-7144-11e5-ba71-058fbc01cf0b	AE	1
ce26e01a-7144-11e5-ba71-058fbc01cf0b	AK	1
ce351572-7144-11e5-ba71-058fbc01cf0b	AK	1
ce3c3b36-7144-11e5-ba71-058fbc01cf0b	AK	1
ce4170ec-7144-11e5-ba71-058fbc01cf0b	AK	1
ce666e38-7144-11e5-ba71-058fbc01cf0b	AK	1
ce7007a4-7144-11e5-ba71-058fbc01cf0b	AK	1
ce718782-7144-11e5-ba71-058fbc01cf0b	AK	1
ce72c64c-7144-11e5-ba71-058fbc01cf0b	AK	1
ce72ef5a-7144-11e5-ba71-058fbc01cf0b	AK	1
ce735210-7144-11e5-ba71-058fbc01cf0b	AK	1
ce741718-7144-11e5-ba71-058fbc01cf0b	AK	1
ce74f656-7144-11e5-ba71-058fbc01cf0b	AK	1
ce756d52-7144-11e5-ba71-058fbc01cf0b	AK	1
ce77149a-7144-11e5-ba71-058fbc01cf0b	AK	1
ce7c50ae-7144-11e5-ba71-058fbc01cf0b	AK	1
ce7cc106-7144-11e5-ba71-058fbc01cf0b	AK	1
ce945e92-7144-11e5-ba71-058fbc01cf0b	AK	1

4. Export your query results to a text file

Next week, we will learn how to complete some basic forms of data analysis in SQL. However, if you know how to use other analysis or visualization software like Excel or Tableau, you can implement these analyses with the SQL skills you have gained already, as long as you can export the results of your SQL queries in a format other software packages can read. Almost every database interface has a different method for exporting query results, so you will need to look up how to do it every time you try a new interface (another place where having a desire to learn new things will come in handy!).

There are two ways to export your query results using our Jupyter interface.

1. You can select and copy the output you see in an output window, and paste it into another program. Although this strategy is very simple, it only works if your output is very limited in size (since you can only paste 1000 rows at a time).
2. You can tell MySQL to put the results of a query into a variable (for our purposes consider a variable to be a temporary holding place), and then use Python code to format the data in the variable as a CSV file (comma separated value file, a .CSV file) that can be downloaded. When you use this strategy, all of the results of a query will be saved into the variable, not just the first 1000 rows as displayed in Jupyter, even if we have set up Jupyter to only display 1000 rows of the output.

Let's see how we could export query results using the second method.

To tell MySQL to put the results of a query into a variable, use the following syntax:

```
    variable_name_of_your_choice = %sql [your full query goes here];
```

In this case, you must execute your SQL query all on one line. So if you wanted to export the list of dog breeds in the dogs table, you could begin by executing:

```
    breed_list = %sql SELECT DISTINCT breed FROM dogs ORDER BY breed;
```

Go ahead and try it:

In [11]:

```
breed_list = %sql SELECT DISTINCT breed FROM dogs ORDER BY breed;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2006 rows affected.
```

Once your variable is created, using the above command tell Jupyter to format the variable as a csv file using the following syntax:

```
    the_output_name_you_want.csv('the_output_name_you_want.csv')
```

Since this line is being run in Python, do NOT include the %sql prefix when trying to execute the line. We could therefore export the breed list by executing:

```
    breed_list.csv('breed_list.csv')
```

When you do this, all of the results of the query will be saved in the text file but the results will not be displayed in your notebook. This is a convenient way to retrieve large amounts of data from a query without taxing your browser or the server.

Try it yourself:

In [10]:

```
breed_list.csv('breed_list.csv')
```

Out[10]:

CSV results (./files/breed_list.csv)

You should see a link in the output line that says "CSV results." You can click on this link to see the text file in a tab in your browser or to download the file to your computer (exactly how this works will differ depending on your browser and settings, but your options will be the same as if you were trying to open or download a file from any other website.)

You can also open the file directly from the home page of your Jupyter account. Behind the scenes, your csv file was written to your directory on the Jupyter server, so you should now see this file listed in your Jupyter account landing page along with the list of your notebooks. Just like a notebook, you can copy it, rename it, or delete it from your directory by clicking on the check box next to the file and clicking the "duplicate," "rename," or trash can buttons at the top of the page.

5. A Bird's Eye View of Other Functions You Might Want to Explore

When you open your breed list results file, you will notice the following:

1) All of the rows of the output are included, even though you can only see 1000 of those rows when you run the query through the Jupyter interface.

2) There are some strange values in the breed list. Some of the entries in the breed column seem to have a dash included before the name. This is an example of what real business data sets look like...they are messy! We will use this as an opportunity to highlight why it is so important to be curious and explore MySQL functions on your own.

If you needed an accurate list of all the dog breeds in the dogs table, you would have to find some way to "clean up" the breed list you just made. Let's examine some of the functions that could help you achieve this cleaning using SQL syntax rather than another program or language outside of the database.

I included these links to MySQL functions in an earlier notebook:
https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html (https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html)
http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

The following description of a function called REPLACE is included in that resource:

"REPLACE(str,from_str,to_str)
Returns the string str with all occurrences of the string from_str replaced by the string to_str. REPLACE() performs a case-sensitive match when searching for from_str."

One thing we could try is using this function to replace any dashes included in the breed names with no character:

```
SELECT DISTINCT breed,
REPLACE(breed,'-','') AS breed_fixed
FROM dogs
ORDER BY breed_fixed
```

In this query, we put the field/column name in the replace function where the syntax instructions listed "str" in order to tell the REPLACE function to act on the entire column. The "-" was the "from_str", which is the string we wanted to replace. The "" was the to_str, which is the character with which we want to replace the "from_str".

Try looking at the output:

In [15]:

```
%%sql
SELECT DISTINCT breed,
REPLACE (breed,'-',' ') AS breed_fixed
FROM dogs
ORDER BY breed_fixed
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[15]:

breed	breed_fixed
-American Eskimo Dog Mix	American Eskimo Dog Mix
-American Pit Bull Terrier Mix	American Pit Bull Terrier Mix
-Anatolian Shepherd Dog Mix	Anatolian Shepherd Dog Mix
-Australian Cattle Dog Mix	Australian Cattle Dog Mix
-Australian Shepherd Mix	Australian Shepherd Mix
-Beagle Mix	Beagle Mix
-Bichon Frise Mix	Bichon Frise Mix
-Bluetick Coonhound Mix	Bluetick Coonhound Mix
-Border Collie Mix	Border Collie Mix
-Boxer Mix	Boxer Mix
-Cairn Terrier Mix	Cairn Terrier Mix
-Cavalier King Charles Spaniel Mix	Cavalier King Charles Spaniel Mix
-Chesapeake Bay Retriever Mix	Chesapeake Bay Retriever Mix
-Chihuahua Mix	Chihuahua Mix
-Cocker Spaniel Mix	Cocker Spaniel Mix
-Collie Mix	Collie Mix
-German Shepherd Dog Mix	German Shepherd Dog Mix
-Golden Retriever Mix	Golden Retriever Mix
-Great Dane Mix	Great Dane Mix
-Great Pyrenees Mix	Great Pyrenees Mix

That was helpful, but you'll still notice some issues with the output.

First, the leading dashes are indeed removed in the breed_fixed column, but now the dashes used to separate breeds in entries like 'French Bulldog-Boston Terrier Mix' are missing as well. So REPLACE isn't the right choice to selectively remove leading dashes.

Perhaps we could try using the TRIM function:

http://www.w3resource.com/mysql/string-functions/mysql-trim-function.php (http://www.w3resource.com/mysql/string-functions/mysql-trim-function.php)

```
SELECT DISTINCT breed, TRIM(LEADING '-' FROM breed) AS breed_fixed
FROM dogs
ORDER BY breed_fixed
```

Try the query written above yourself, and inspect the output carefully:

In [3]:

```
%%sql
SELECT DISTINCT breed, TRIM(LEADING '-' FROM breed) AS breed_fixed
FROM dogs
ORDER BY breed_fixed
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[3]:

breed	breed_fixed
Affenpinscher	Affenpinscher
Affenpinscher-Afghan Hound Mix	Affenpinscher-Afghan Hound Mix
Affenpinscher-Airedale Terrier Mix	Affenpinscher-Airedale Terrier Mix
Affenpinscher-Alaskan Malamute Mix	Affenpinscher-Alaskan Malamute Mix
Affenpinscher-American English Coonhound Mix	Affenpinscher-American English Coonhound Mix
Affenpinscher-Brussels Griffon Mix	Affenpinscher-Brussels Griffon Mix
Affenpinscher-Poodle Mix	Affenpinscher-Poodle Mix
Affenpinscher-Rat Terrier Mix	Affenpinscher-Rat Terrier Mix
Affenpinscher-Spanish Water Dog Mix	Affenpinscher-Spanish Water Dog Mix
Afghan Hound	Afghan Hound
Afghan Hound-Affenpinscher Mix	Afghan Hound-Affenpinscher Mix
Afghan Hound-Airedale Terrier Mix	Afghan Hound-Airedale Terrier Mix
Afghan Hound-Akita Mix	Afghan Hound-Akita Mix
Afghan Hound-Bloodhound Mix	Afghan Hound-Bloodhound Mix
Afghan Hound-Border Collie Mix	Afghan Hound-Border Collie Mix
Afghan Hound-Golden Retriever Mix	Afghan Hound-Golden Retriever Mix
Airedale Terrier	Airedale Terrier
Airedale Terrier-Afghan Hound Mix	Airedale Terrier-Afghan Hound Mix
Airedale Terrier-Border Collie Mix	Airedale Terrier-Border Collie Mix
Airedale Terrier-Catahoula Leopard Dog Mix	Airedale Terrier-Catahoula Leopard Dog Mix

That certainly gets us a lot closer to the list we might want, but there are still some entries in the breed_fixed column that are conceptual duplicates of each other, due to poor consistency in how the breed names were entered. For example, one entry is "Beagle Mix" while another is "Beagle- Mix". These entries are clearly meant to refer to the same breed, but they will be counted as separate breeds as long as their breed names are different.

Cleaning up all of the entries in the breed column would take quite a bit of work, so we won't go through more details about how to do it in this lesson. Instead, use this exercise as a reminder for why it's so important to always look at the details of your data, and as motivation to explore the MySQL functions we won't have time to discuss in the course. If you push yourself to learn new SQL functions and embrace the habit of getting to know your data by exploring its raw values and outputs, you will find that SQL provides very efficient tools to clean real-world messy data sets, and you will arrive at the correct conclusions about what your data indicate your company should do.

Now it's time to practice using AS, DISTINCT, and ORDER BY in your own queries.

Question 4: How would you get a list of all the subcategories of Dognition tests, in alphabetical order, with no test listed more than once (if you do not limit your output, you should retrieve 16 rows)?

In [5]:

```
%%sql
SELECT DISTINCT subcategory_name
FROM complete_tests
ORDER BY subcategory_name ASC
LIMIT 16;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
16 rows affected.
```

Out[5]:

subcategory_name
Communication
Cunning
Empathy
Expression Game
Impossible Task
Laterality
Memory
Numerosity
Perspective Game
Reasoning
Self Control Game
Shaker Game
Shell Game
Smell Game
Social Bias
Spatial Navigation

Question 5: How would you create a text file with a list of all the non-United States countries of Dognition customers with no country listed more than once?

In [12]:

```
non_US_countries = %sql SELECT DISTINCT user_guid, country FROM users WHERE country!='US';
non_US_countries.csv('non_US_countries.csv')
```

```
 * mysql://studentuser:***@localhost/dognitiondb
6905 rows affected.
```

Out[12]:

CSV results (./files/non_US_countries.csv)

Question 6: How would you find the User ID, Dog ID, and test name of the first 10 tests to ever be completed in the Dognition database?

In [16]:

```
%%sql
SELECT DISTINCT user_guid, dog_guid, test_name
FROM complete_tests
ORDER BY created_at
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[16]:

user_guid	dog_guid	test_name
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Yawn Warm-up
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Yawn Game
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Eye Contact Warm-up
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Eye Contact Game
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Treat Warm-up
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Arm Pointing
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Foot Pointing
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Watching
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Turn Your Back
None	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Cover Your Eyes

Question 7: How would create a text file with a list of all the customers with yearly memberships who live in the state of North Carolina (USA) and joined Dognition after March 1, 2014, sorted so that the most recent member is at the top of the list?

In [18]:

```
NC_Yearly_after_March_1_2014 = %sql SELECT DISTINCT user_guid, state, created_at FROM users WHERE membership_type
= 2 and state = 'NC' and country = 'US' and created_at > '2014-03,01' ORDER BY created_at DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
68 rows affected.
```

In [19]:

```
NC_Yearly_after_March_1_2014.csv('NC_Yearly_after_March_1_2014.csv')
```

Out[19]:

CSV results (./files/NC_Yearly_after_March_1_2014.csv)

Question 8: See if you can find an SQL function from the list provided at:

http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

that would allow you to output all of the distinct breed names in UPPER case. Create a query that would output a list of these names in upper case, sorted in alphabetical order.

In [23]:

```
%%sql
SELECT DISTINCT UPPER(breed)
FROM dogs
ORDER BY breed
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[23]:

UPPER(breed)
-AMERICAN ESKIMO DOG MIX
-AMERICAN PIT BULL TERRIER MIX
-ANATOLIAN SHEPHERD DOG MIX
-AUSTRALIAN CATTLE DOG MIX
-AUSTRALIAN SHEPHERD MIX
-BEAGLE MIX
-BICHON FRISE MIX
-BLUETICK COONHOUND MIX
-BORDER COLLIE MIX
-BOXER MIX
-CAIRN TERRIER MIX
-CAVALIER KING CHARLES SPANIEL MIX
-CHESAPEAKE BAY RETRIEVER MIX
-CHIHUAHUA MIX
-COCKER SPANIEL MIX
-COLLIE MIX
-GERMAN SHEPHERD DOG MIX
-GOLDEN RETRIEVER MIX
-GREAT DANE MIX
-GREAT PYRENEES MIX

Practice any other queries you want to try below!

In [ ]:

```

```

MySQL Exercise 4: Summarizing your Data

Last week you practiced retrieving and formatting selected subsets of raw data from individual tables in a database. In this lesson we are going to learn how to use SQL to run calculations that summarize your data without having to output all the raw rows or entries. These calculations will serve as building blocks for the queries that will address our business questions about how to improve Dognition test completion rates.

These are the five most common aggregate functions used to summarize information stored in tables:

Function	What it Returns
AVG()	column average
COUNT()	# of rows in a column
MAX()	a column's highest value
MIN()	a column's lowest value
SUM()	sum of all a column's values

You will use COUNT and SUM very frequently.

COUNT is the only aggregate function that can work on any type of variable. The other four aggregate functions are only appropriate for numerical data.

All aggregate functions require you to enter either a column name or a "*" in the parentheses after the function word.

Let's begin by exploring the COUNT function.

1. The COUNT function

First, load the sql library and the Dognition database, and set dognition as the default database.

In [2]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[2]:

```
[]
```

The Jupyter interface conveniently tells us how many rows are in our query output, so we can compare the results of the COUNT function to the results of our SELECT function. If you run:

```
SELECT breed
FROM dogs
```

Jupyter tells that 35050 rows are "affected", meaning there are 35050 rows in the output of the query (although, of course, we have limited the display to only 1000 rows at a time).

Now try running:

```
SELECT COUNT(breed)
FROM dogs
```

In [6]:

```
%%sql
SELECT COUNT(breed)
FROM dogs
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[6]:

COUNT(breed)
35050

COUNT is reporting how many rows are in the breed column in total. COUNT should give you the same output as Jupyter's output without displaying the actual rows of data that are being aggregated.

You can use DISTINCT (which you learned about in MySQL Exercise 3) with COUNT to count all the unique values in a column, but it must be placed inside the parentheses, immediately before the column that is being counted. For example, to count the number of distinct breed names contained within all the entries in the breed column you could query:

```
SELECT COUNT(DISTINCT breed)
  FROM dogs
```

What if you wanted to know how many indivdual dogs successfully completed at least one test?

Since every row in the complete_tests table represents a completed test and we learned earlier that there are no NULL values in the created_at column of the complete_tests table, any non-null Dog_Guid in the complete_tests table will have completed at least one test. When a column is included in the parentheses, null values are automatically ignored. Therefore, you could use:

```
SELECT COUNT(DISTINCT Dog_Guid)
  FROM complete_tests
```

Question 1: Try combining this query with a WHERE clause to find how many individual dogs completed tests after March 1, 2014 (the answer should be 13,289):

In [15]:

```
%%sql
SELECT COUNT(DISTINCT Dog_Guid)
FROM complete_tests
WHERE created_at>'2014_03_01';
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[15]:

COUNT(DISTINCT Dog_Guid)
13289

You can use the "*" in the parentheses of a COUNT function to count how many rows are in the entire table (or subtable). There are two fundamental difference between COUNT(*) and COUNT(column_name), though.

The first difference is that you cannot use DISTINCT with COUNT(*).

Question 2: To observe the second difference yourself first, count the number of rows in the dogs table using COUNT(*):

In [16]:

```
%%sql
SELECT COUNT(*)
FROM dogs
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[16]:

COUNT(*)
35050

Question 3: Now count the number of rows in the exclude column of the dogs table:

In [17]:

```
%%sql
SELECT COUNT(exclude)
FROM dogs
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[17]:

COUNT(exclude)
1025

The output of the second query should return a much smaller number than the output of the first query. That's because:

> When a column is included in a count function, null values are ignored in the count. When an asterisk is included in a count function, nulls are included in the count.

This will be both useful and important to remember in future queries where you might want to use SELECT(*) to count items in multiple groups at once.

Question 4: How many distinct dogs have an exclude flag in the dogs table (value will be "1")? (the answer should be 853)

In [18]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM dogs
WHERE exclude=1;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[18]:

COUNT(DISTINCT dog_guid)
853

2. The SUM Function

The fact that the output of:

```
SELECT COUNT(exclude)
FROM dogs
```

was so much lower than:

```
SELECT COUNT(*)
FROM dogs
```

suggests that there must be many NULL values in the exclude column. Conveniently, we can combine the SUM function with ISNULL to count exactly how many NULL values there are. Look up "ISNULL" at this link to MySQL functions I included in an earlier lesson:

http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

You will see that ISNULL is a logical function that returns a 1 for every row that has a NULL value in the specified column, and a 0 for everything else. If we sum up the number of 1s outputted by ISNULL(exclude), then, we should get the total number of NULL values in the column. Here's what that query would look like:

```
SELECT SUM(ISNULL(exclude))
FROM dogs
```

It might be tempting to treat SQL like a calculator and leave out the SELECT statement, but you will quickly see that doesn't work.

> *Every SQL query that extracts data from a database MUST contain a SELECT statement.* <mark>

Try counting the number of NULL values in the exclude column:

In [20]:

```
%%sql
SELECT SUM(ISNULL(exclude))
FROM dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[20]:

SUM(ISNULL(exclude))
34025

The output should return a value of 34,025. When you add that number to the 1025 entries that have an exclude flag, you get a total of 35,050, which is the number of rows reported by SELECT COUNT(*) from dogs.

3. The AVG, MIN, and MAX Functions

AVG, MIN, and MAX all work very similarly to SUM.

During the Dognition test, customers were asked the question: "How surprising were [your dog's name]'s choices?" after completing a test. Users could choose any number between 1 (not surprising) to 9 (very surprising). We could retrieve the average, minimum, and maximum rating customers gave to this question after completing the "Eye Contact Game" with the following query:

```
SELECT test_name,
AVG(rating) AS AVG_Rating,
MIN(rating) AS MIN_Rating,
MAX(rating) AS MAX_Rating
FROM reviews
WHERE test_name="Eye Contact Game";
```

This would give us an output with 4 columns. The last three columns would have titles reflecting the names inputted after the AS clauses. Recall that if you want to title a column with a string of text that contains a space, that string will need to be enclosed in quotation marks after the AS clause in your query.

Question 5: What is the average, minimum, and maximum ratings given to "Memory versus Pointing" game? (Your answer should be 3.5584, 0, and 9, respectively)

In [3]:

```
%%sql
SELECT test_name,
AVG(rating) AS AVG_rating,
MIN(rating) AS MIN_rating,
MAX(rating) AS MAX_rating
FROM reviews
WHERE test_name= "Memory_versus_Pointing";
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[3]:

test_name	AVG_rating	MIN_rating	MAX_rating
None	None	None	None

What if you wanted the average rating for each of the 40 tests in the Reviews table? One way to do that with the tools you know already is to write 40 separate queries like the ones you wrote above for each test, and then copy or transcribe the results into a separate table in another program like Excel to assemble all the results in one place. That would be a very tedious and time-consuming exercise. Fortunately, there is a very simple way to produce the results you want within one query. That's what we will learn how to do in MySQL Exercise 5. However, it is important that you feel comfortable with the syntax we have learned thus far before we start taking advantage of that functionality. Practice is the best way to become comfortable!

Practice incorporating aggregate functions with everything else you've learned so far in your own queries.

Question 6: How would you query how much time it took to complete each test provided in the exam_answers table, in minutes? Title the column that represents this data "Duration." Note that the exam_answers table has over 2 million rows, so if you don't limit your output, it will take longer than usual to run this query. (HINT: use the TIMESTAMPDIFF function described at: http://www.w3resource.com/mysql/date-and-time-functions/date-and-time-functions.php (http://www.w3resource.com/mysql/date-and-time-functions/date-and-time-functions.php). It might seem unkind of me to keep suggesting you look up and use new functions I haven't demonstrated for you, but I really want you to become confident that you know how to look up and use new functions when you need them! It will give you a very competitive edge in the business world.)

In [8]:

```
%%sql
SELECT TIMESTAMPDIFF(minute,start_time,end_time) AS Duration
FROM exam_answers
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[8]:

Duration
345139
345139
345139
345138
345138
345138
345138
345138
345137
345137

Question 7: Include a column for Dog_Guid, start_time, and end_time in your query, and examine the output. Do you notice anything strange?

In [10]:

```
%%sql
SELECT dog_guid, start_time,end_time, TIMESTAMPDIFF(minute,start_time,end_time) AS Duration
FROM exam_answers
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[10]:

dog_guid	start_time	end_time	Duration
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 03:58:13	2013-10-02 20:18:06	345139
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 03:58:31	2013-10-02 20:18:06	345139
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 03:59:03	2013-10-02 20:18:06	345139
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 03:59:10	2013-10-02 20:18:06	345138
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 03:59:22	2013-10-02 20:18:06	345138
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 03:59:36	2013-10-02 20:18:06	345138
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 03:59:41	2013-10-02 20:18:06	345138
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 04:00:00	2013-10-02 20:18:06	345138
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 04:00:16	2013-10-02 20:18:06	345137
fd27b272-7144-11e5-ba71-058fbc01cf0b	2013-02-05 04:00:35	2013-10-02 20:18:06	345137

If you explore your output you will find that some of your calculated durations appear to be "0." In some cases, you will see many entries from the same Dog_ID with the same start time and end time. That should be impossible. These types of entries probably represent tests run by the Dognition team rather than real customer data. In other cases, though, a "0" is entered in the Duration column even though the start_time and end_time are different. This is because we instructed the function to output the time difference in minutes; unless you change your settings, it will output "0" for any time differences less than the integer 1. If you change your function to output the time difference in seconds, the duration in most of these columns will have a non-zero number.

Question 8: What is the average amount of time it took customers to complete all of the tests in the exam_answers table, if you do not exclude any data (the answer will be approximately 587 minutes)?

In [12]:

```
%%sql
SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
FROM exam_answers;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[12]:

AvgDuration
586.9041

Question 9: What is the average amount of time it took customers to complete the "Treat Warm-Up" test, according to the exam_answers table (about 165 minutes, if no data is excluded)?

In [15]:

```
%%sql
SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
FROM exam_answers
WHERE test_name="Treat_Warm-Up";
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[15]:

AvgDuration
None

Question 10: How many possible test names are there in the exam_answers table?

In [22]:

```
%%sql
SELECT DISTINCT test_name
FROM exam_answers
ORDER BY test_name;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
68 rows affected.
```

Out[22]:

test_name
None
1 vs 1 Game
3 vs 1 Game
5 vs 1 Game
Activity
Arm Pointing
Attachment
Confinement
Cover Your Eyes
Delayed Cup Game
Diet
Different Perspective
Emotions
Environment
Excitability
Expression Game
Eye Contact Game
Eye Contact Warm-up
Foot Pointing
Gender
Impossible Task Game
Impossible Task Warm-up
Inferential Reasoning Game

Inferential Reasoning Game

Inferential Reasoning Warm-up

Memory versus Pointing

Memory versus Smell

Navigation Game

Navigation Learning

Navigation Warm-up

Numerosity Warm-Up

Obedience

One Cup Warm-up

Owner

Partnership

Perception

Physical

Physical Reasoning Game

Physical Reasoning Warm-up

Purina

Purina-Only

Puzzles

Recall

Self Control Game

Set 1

Set 2

Set 3

Shaker Game

Shaker Warm-Up

Shared Perspective

Shy/Boldness

Slide

Smell Game

Sociability

Social

Social-Quiz

Stair Game

Surprise And Delight

Switch

Toys

Training

Treat Warm-up

Turn Your Back

Two Cup Warm-up

Warm-Up

Watching

Watching - Part 2

Yawn Game

Yawn Warm-up

In [27]:

```
%%sql
SELECT COUNT(DISTINCT test_name)
FROM exam_answers;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[27]:

COUNT(DISTINCT test_name)
67

In [24]:

```
%%sql
SELECT DISTINCT test_name
FROM complete_tests
ORDER BY test_name;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
40 rows affected.
```

Out[24]:

test_name
1 vs 1 Game
3 vs 1 Game
5 vs 1 Game
Arm Pointing
Cover Your Eyes
Delayed Cup Game
Different Perspective
Expression Game
Eye Contact Game
Eye Contact Warm-up
Foot Pointing
Impossible Task Game
Impossible Task Warm-up
Inferential Reasoning Game
Inferential Reasoning Warm-up
Memory versus Pointing
Memory versus Smell
Navigation Game
Navigation Learning
Navigation Warm-up
Numerosity Warm-Up
One Cup Warm-up
Physical Reasoning Game
Physical Reasoning Warm-up
Self Control Game
Shaker Game
Shaker Warm-Up
Shared Perspective
Slide
Smell Game
Stair Game
Switch
Treat Warm-up
Turn Your Back
Two Cup Warm-up
Warm-Up
Watching
Watching - Part 2
Yawn Game
Yawn Warm-up

In [28]:

```
%%sql
SELECT COUNT(DISTINCT test_name)
FROM complete_tests;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[28]:

COUNT(DISTINCT test_name)
40

You should have discovered that the exam_answers table has many more test names than the completed_tests table. It turns out that this table has information about experimental tests that Dognition has not yet made available to its customers.

Question 11: What is the minimum and maximum value in the Duration column of your query that included the data from the entire table?

In [26]:

```
%%sql
SELECT MIN(TIMESTAMPDIFF(minute,start_time,end_time)) AS MinDuration,
MAX(TIMESTAMPDIFF(minute,start_time,end_time)) AS MaxDuration
FROM exam_answers;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[26]:

MinDuration	MaxDuration
-187	1036673

The minimum Duration value is *negative*! The end_times entered in rows with negative Duration values are earlier than the start_times. Unless Dognition has created a time machine, that's impossible and these entries must be mistakes.

Question 12: How many of these negative Duration entries are there? (the answer should be 620)

In [29]:

```
%%sql
SELECT COUNT(TIMESTAMPDIFF(minute,start_time,end_time)) AS Duration
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time) < 0;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[29]:

Duration
620

Question 13: How would you query all the columns of all the rows that have negative durations so that you could examine whether they share any features that might give you clues about what caused the entry mistake?

In [33]:

```
%%sql
SELECT *
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time) < 0
LIMIT 15;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
15 rows affected.
```

Out[33]:

script_detail_id	subcategory_name	test_name	step_type	start_time	end_time	loop_number	dog_guid
60	Empathy	Eye Contact Warm-up	question	2013-02-17 20:35:43	2013-02-17 20:34:43	3	fd3fe18a-7144-11e5-ba71-058fbc01cf0b
558	Sociability	Sociability	question	2013-02-18 04:25:19	2013-02-18 04:24:18	0	fd3fe50e-7144-11e5-ba71-058fbc01cf0b
557	Sociability	Sociability	question	2013-02-18 07:44:09	2013-02-18 07:43:09	0	fd3fe5ea-7144-11e5-ba71-058fbc01cf0b
574	Shy/Boldness	Shy/Boldness	question	2013-02-18 07:46:14	2013-02-18 07:45:13	0	fd3fe5ea-7144-11e5-ba71-058fbc01cf0b
582	Shy/Boldness	Shy/Boldness	question	2013-02-18 07:47:07	2013-02-18 07:46:06	0	fd3fe5ea-7144-11e5-ba71-058fbc01cf0b
600	Sociability	Sociability	question	2013-02-18 07:50:07	2013-02-18 07:49:07	0	fd3fe5ea-7144-11e5-ba71-058fbc01cf0b
293	Memory	Two Cup Warm-up	question	2013-02-18 13:23:25	2013-02-18 13:22:23	2	fd3fbd7c-7144-11e5-ba71-058fbc01cf0b
293	Memory	Two Cup Warm-up	question	2013-02-18 13:23:31	2013-02-18 13:22:28	4	fd3fbd7c-7144-11e5-ba71-058fbc01cf0b
322	Memory	Memory versus Pointing	question	2013-02-18 13:25:15	2013-02-18 13:24:14	1	fd3fbd7c-7144-11e5-ba71-058fbc01cf0b
322	Memory	Memory versus Pointing	question	2013-02-18 13:25:30	2013-02-18 13:24:27	5	fd3fbd7c-7144-11e5-ba71-058fbc01cf0b
352	Memory	Memory versus Smell	question	2013-02-18 13:27:55	2013-02-18 13:26:53	4	fd3fbd7c-7144-11e5-ba71-058fbc01cf0b
442	Reasoning	Inferential Reasoning Warm-up	question	2013-02-18 15:49:19	2013-02-18 15:48:18	5	fd3d4952-7144-11e5-ba71-058fbc01cf0b
442	Reasoning	Inferential Reasoning Warm-up	question	2013-02-18 15:50:15	2013-02-18 15:49:14	6	fd3d4952-7144-11e5-ba71-058fbc01cf0b
475	Reasoning	Inferential Reasoning Game	question	2013-02-18 15:53:47	2013-02-18 15:52:46	1	fd3d4952-7144-11e5-ba71-058fbc01cf0b
475	Reasoning	Inferential Reasoning Game	question	2013-02-18 15:54:40	2013-02-18 15:53:39	2	fd3d4952-7144-11e5-ba71-058fbc01cf0b

Question 14: What is the average amount of time it took customers to complete all of the tests in the exam_answers table when the negative durations are excluded from your calculation (you should get 11233 minutes)?

In [32]:

```
%%sql
SELECT AVG(TIMESTAMPDIFF(minute,start_time,end_time)) AS AvgDuration
FROM exam_answers
WHERE TIMESTAMPDIFF(minute,start_time,end_time) > 0;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[32]:

AvgDuration
11233.0951

You have just seen another first-hand example of how messy real-world data can be, and how easy it can be to miss the "mess" when your data sets are too large to examine thoroughly by eye. Before continuing on to the next SQL lesson, make sure to watch the video about how you as a data analyst can practice building habits that will prevent you from being fooled by messy data.

And, as always, feel free to practice more queries here!

In [ ]:

```
```

MySQL Exercise 6: Common Pitfalls of Grouped Queries

There are two main reasons grouped queries can cause problems, especially in MySQL:

> 1) MySQL gives the user the benefit of the doubt, and assumes we don't make (at least some kinds of) mistakes. Unfortunately, we do make those mistakes.
>
> 2) We commonly think about data as spreadsheets that allow you make calculations across rows and columns, and that allow you to keep both raw and aggregated data in the same spreadsheet. Relational databases don't work that way.

The way these issues cause problems are:

> 1) When we are working with a MySQL database, we incorrectly interpret non-sensical output from illogical queries, or
>
> 2) When we are working with a non-MySQL database platform, we struggle with trying to make queries that will never work because they ask for both aggregated and non-aggregated data.

In this lesson, we will learn what these issues look like.

1. Misinterpretations due to Aggregation Mismatches

Begin by loading the SQL library, connecting to the Dognition database, and setting the Dognition database as the default.

In [2]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[2]:

```
[]
```

Imagine that we would like to retrieve, for each breed_type in the Dognition database, the number of unique dog_guids associated with that breed_type and their weight. Let's try to write a query that reflects that request:

```
SELECT breed_type, COUNT(DISTINCT dog_guid) AS NumDogs, weight
FROM dogs
GROUP BY breed_type;
```

Now take a look at the output:

In [3]:

```
%%sql
SELECT breed_type, COUNT(DISTINCT dog_guid) AS NumDogs, weight
FROM dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[3]:

breed_type	NumDogs	weight
Pure Breed	35050	50

In [7]:

```
%%sql
SELECT DISTINCT breed_type, weight
FROM dogs
ORDER BY breed_type ASC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
76 rows affected.
```

Out[7]:

breed_type	weight
Cross Breed	0
Cross Breed	10
Cross Breed	20
Cross Breed	30
Cross Breed	40
Cross Breed	50
Cross Breed	60
Cross Breed	70
Cross Breed	80
Cross Breed	90
Cross Breed	100
Cross Breed	110
Cross Breed	120
Cross Breed	130
Cross Breed	140
Cross Breed	150
Cross Breed	160
Cross Breed	170
Cross Breed	180
Cross Breed	190
Cross Breed	250
Mixed Breed/ Other/ I Don't Know	0
Mixed Breed/ Other/ I Don't Know	10
Mixed Breed/ Other/ I Don't Know	20
Mixed Breed/ Other/ I Don't Know	30
Mixed Breed/ Other/ I Don't Know	40
Mixed Breed/ Other/ I Don't Know	50
Mixed Breed/ Other/ I Don't Know	60
Mixed Breed/ Other/ I Don't Know	70
Mixed Breed/ Other/ I Don't Know	80
Mixed Breed/ Other/ I Don't Know	90
Mixed Breed/ Other/ I Don't Know	100
Mixed Breed/ Other/ I Don't Know	110
Mixed Breed/ Other/ I Don't Know	120
Mixed Breed/ Other/ I Don't Know	130
Mixed Breed/ Other/ I Don't Know	140
Mixed Breed/ Other/ I Don't Know	150
Mixed Breed/ Other/ I Don't Know	160
Mixed Breed/ Other/ I Don't Know	170
Mixed Breed/ Other/ I Don't Know	190
Popular Hybrid	0
Popular Hybrid	10
Popular Hybrid	20
Popular Hybrid	30
Popular Hybrid	40
Popular Hybrid	50
Popular Hybrid	60
Popular Hybrid	70
Popular Hybrid	80
Popular Hybrid	90
Popular Hybrid	100
Popular Hybrid	110
Popular Hybrid	120
Popular Hybrid	130

Popular Hybrid	140
Popular Hybrid	160
Pure Breed	0
Pure Breed	10
Pure Breed	20
Pure Breed	30
Pure Breed	40
Pure Breed	50
Pure Breed	60
Pure Breed	70
Pure Breed	80
Pure Breed	90
Pure Breed	100
Pure Breed	110
Pure Breed	120
Pure Breed	130
Pure Breed	140
Pure Breed	150
Pure Breed	160
Pure Breed	170
Pure Breed	180
Pure Breed	190

In [8]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid) AS NumDogs
FROM dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[8]:

NumDogs
35050

In [6]:

```
%%sql
SELECT DISTINCT breed_type
FROM dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[6]:

breed_type
Pure Breed
Mixed Breed/ Other/ I Don't Know
Cross Breed
Popular Hybrid

In [4]:

```
%%sql
SELECT breed_type, COUNT(DISTINCT dog_guid) AS NumDogs, weight
FROM dogs
GROUP BY breed_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[4]:

breed_type	NumDogs	weight
Cross Breed	5568	0
Mixed Breed/ Other/ I Don't Know	9499	50
Popular Hybrid	1160	70
Pure Breed	18823	50

You immediately notice a few things: (1) the query accurately represents the fields I said I wanted; (2) the query executed without errors! Wonderful! (3) Cross Breed dogs weigh 0 pounds; and (4) the grammar of the sentence describing what I said I wanted seems a little confusing: "We would like to retrieve, for *each breed_type* in the Dognition database, *the number of* unique dog_guids associated with that breed_type and *their weight*."

All of these things you noticed are related. Let's address them in reverse order.

What's wrong with the sentence I wrote? One of the things I said I wanted was *the number of* unique dog_guids. This is a single number. I also said I wanted "their weight." "Their" implies many weight measurements, not one measurement. In order to make my grammar correct, I need my description of dog_guids and weight to either both be singular or both be plural. To make the logic behind the sentence make sense, I have to do a similar thing: either dog_guids and weight both need to be aggregated or dog_guids and weight both need to be non-aggregated.

It's useful to remember that SQL output is always a table. How could you construct a valid table that would have columns for aggregate counts and individual weight measurements at the same time? The answer is, you can't. One option is to disaggregate the count so that you have one column with dog_guids and another column with weight measurements for each dog_guid. The only other option is to aggregate the weight measurements so that you have one column with the total count of dog_guids and another column with the average (or some other kind of summary aggregation) weight measurement for the group the count represents.

That brings us to the next phenomenon we observed: Cross Breed dogs weigh 0 pounds. Well, unless the laws of gravity and physics have changed, that's not possible. Something strange must be happening in the weight field.

We've established that the question I posed and the query I executed don't make logical sense, yet the MySQL query did run! If there is no way to make a tablular output that fits what I asked for, what is MySQL outputting for us?

It turns out that MySQL resolves my poor query by choosing its own way to "summarize" the unaggregated field, which in our case is "weight." Rather than run an aggregation function that takes all the values in the weight column into account, though, it unpredictably populates the weight output column with one value from all the possible weight values within a given breed_type subset. Which value it chooses will be different depending on the original order of the raw data and the configuration of a database. This flexibility is very convenient when you know that all the values in a non-aggregated column are the same for the subsets of the data that correspond to the variable by which you are grouping. In fact, the visualization software Tableau (which is based in SQL language) recognized how frequently this type of situation arises and came up with a custom solution for its customers. Tableau incorprated an aggregation-like function called "ATTR" into its interface to let users say "I'm using an aggregation function here because SQL says I have to, but I know that this is a situation where all of the rows in each group will have the same value."

Tableau's approach is helpful because it forces users to acknowledge that a field in a query is supposed to be aggregated, and Tableau's formulas will crash if all the rows in a group do not have the same value. MySQL doesn't force users to do this. MySQL trusts users to know what they are doing, and will provide an output even if all the rows in a group do not have the same value. Unfortunately, this approach can cause havoc if you aren't aware of what you are asking MySQL to do and aren't familiar with your data.

Let's see a couple more first-hand examples of this tricky GROUP BY behavior. Let's assume you want to know the number of each kind of test completed in different months of the year.

You execute the following query:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY test_name
ORDER BY test_name ASC, Month ASC;
```

Question 1: What does the Month column represent in this output? Take a look and see what you think:

In [9]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY test_name
ORDER BY test_name ASC, Month ASC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
40 rows affected.
```

Out[9]:

test_name	Month	Num_Completed_Tests
1 vs 1 Game	6	255
3 vs 1 Game	5	368
5 vs 1 Game	5	620
Arm Pointing	2	11452
Cover Your Eyes	2	7250
Delayed Cup Game	2	5271
Different Perspective	11	89
Expression Game	10	124
Eye Contact Game	2	14545
Eye Contact Warm-up	2	16238
Foot Pointing	2	9751
Impossible Task Game	4	532
Impossible Task Warm-up	4	539
Inferential Reasoning Game	2	4980
Inferential Reasoning Warm-up	2	5098
Memory versus Pointing	2	6525
Memory versus Smell	2	6163
Navigation Game	3	927
Navigation Learning	3	1054
Navigation Warm-up	3	1299
Numerosity Warm-Up	5	382
One Cup Warm-up	2	6785
Physical Reasoning Game	2	4255
Physical Reasoning Warm-up	2	4761
Self Control Game	9	132
Shaker Game	2	77
Shaker Warm-Up	2	81
Shared Perspective	11	83
Slide	8	147
Smell Game	7	168
Stair Game	12	112
Switch	8	137
Treat Warm-up	2	11737
Turn Your Back	2	7428
Two Cup Warm-up	2	6681
Warm-Up	6	873
Watching	2	8808
Watching - Part 2	2	7044
Yawn Game	2	19612
Yawn Warm-up	2	20863

Now try a similar query, but GROUP BY Month instead of test_name:

```
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY Month
ORDER BY Month ASC, test_name ASC;
```

Question 2: What does test_name mean in this case? Try it out:

In [10]:

```
%%sql
SELECT test_name, MONTH(created_at) AS Month, COUNT(created_at) AS Num_Completed_Tests
FROM complete_tests
GROUP BY Month
ORDER BY test_name ASC, Month ASC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
12 rows affected.
```

Out[10]:

test_name	Month	Num_Completed_Tests
Delayed Cup Game	1	11068
Delayed Cup Game	5	21013
Eye Contact Game	7	15977
Foot Pointing	6	23381
Inferential Reasoning Warm-up	11	12652
Inferential Reasoning Warm-up	12	10859
Memory versus Smell	8	13382
Physical Reasoning Game	4	7130
Yawn Warm-up	2	9122
Yawn Warm-up	3	9572
Yawn Warm-up	9	19853
Yawn Warm-up	10	39237

It looks like in both of these cases, MySQL is likely populating the unaggregated column with the first value it finds in that column within the first "group" of rows it is examining.

So how do we prevent this from happening?

> The only way to be sure how the MySQL database will summarize a set of data in a SELECT clause is to tell it how to do so with an aggregate function.<mark>

I should have written my original request to read:

"I would like to know, for *each breed type* of dog, *the number of* unique Dog_Guids there are in the Dognition database and *the breed_type's average weight*."

The query that would have reflected this sentence would have executed an aggregate function for both Dog_Guids and weight. The output of these aggregate functions would be unambiguous, and would easily be represented in a single table.

2. Errors due to Aggregation Mismatches

It is important to note that the issues I described above are the consequence of mismatching aggregate and non-aggregate functions through the GROUP BY clause in MySQL, but other databases manifest the problem in a different way. Other databases won't allow you to run the queries described above at all. When you try to do so, you get an error message that sounds something like:

```
Column 'X' is invalid in the select list because it is not contained in either an aggregate function or th
e GROUP BY clause.
```

Especially when you are just starting to learn MySQL, these error messages can be confusing and infuriating. A good discussion of this problem can be found here:

http://weblogs.sqlteam.com/jeffs/archive/2007/07/20/but-why-must-that-column-be-contained-in-an-aggregate.aspx (http://weblogs.sqlteam.com/jeffs/archive/2007/07/20/but-why-must-that-column-be-contained-in-an-aggregate.aspx)

As a way to prevent these logical mismatches or error messages, you will often hear a rule that "every non-aggregated field that is listed in the SELECT list *must* be listed in the GROUP BY list." You have just seen that this rule is not true in MySQL, which makes MySQL both more flexible and more tricky to work with. However, it is a useful rule of thumb for helping you avoid unknown mismatch errors.

3. By the way, even if you want to, there is no way to intentionally include aggregation mismatches in a single query

You might want to know the total number of unique User_Guids in the Dognition database, and in addition, the total number of unique User_Guids and average weight associated with each breed type. Given that you want to see the information efficiently to help you make decisions, you would like all of this information in one output. After all, that would be easy to do in Excel, given that all of this information could easily be summarized in a single worksheet.

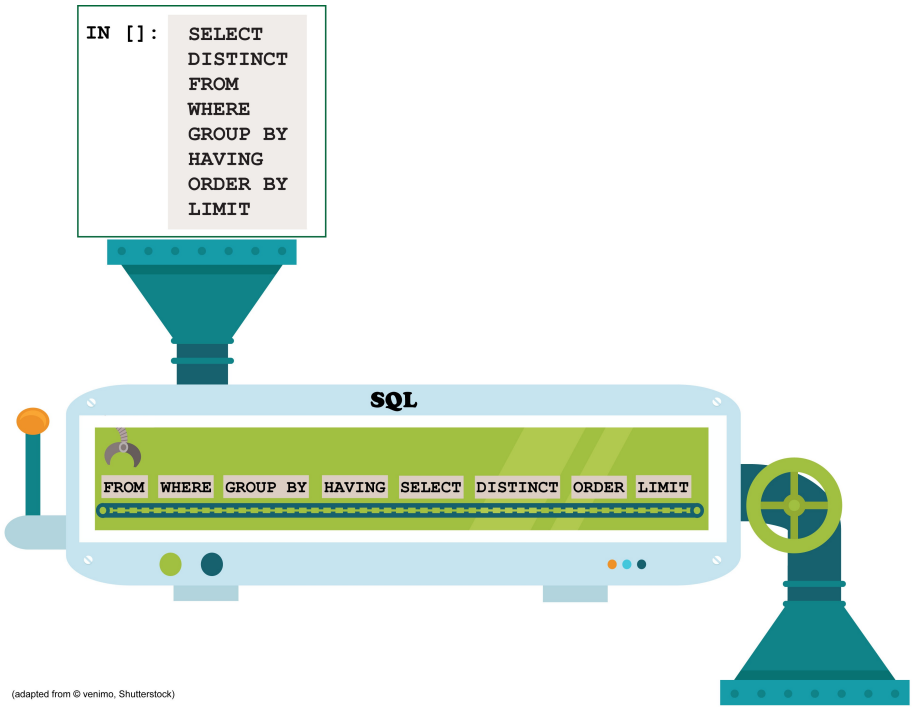
To retrieve this information, you try one of the queries described above. Since you know the rule describing the relationship between fields in the SELECT and GROUP BY clauses, you write:

```
SELECT COUNT(DISTINCT dog_guid), breed_type, AVG(weight) AS avg_weight,
FROM dogs
GROUP BY breed_type;
```

The output to your query gives you four rows with the correct information, but it doesn't give you a count of the entire table without the groups being applied. Surely there must be a way to write a sophisticated query that can put these two pieces of information together for you, right?

Hopefully the discussion in the section above has already made it clear that the answer to this has to be "no." The output of every SQL query is a table. Can you think of a single table that could logically contain aggregated and non-aggregated data? You could put both types of information in an Excel worksheet, but not a single table.

There's yet another more practical reason the information you want can't be selected in a single query. The order of SQL queries is meant to reflect the way we write sentences, but in actuality they are actually executed in a different order than we write them. The cartoon below shows the order we write the queries being sent to the database at the top of the funnel, and the order the database usually executes the queries on the conveyer belt.



This diagram shows you that data are actually grouped before the SELECT expressions are applied. That means that when a GROUP BY expression is included in an SQL query, there is no way to use a SELECT statement to summarize data that cross multiple groups. The data will have already been separated by the time the SELECT statement is applied. The only way to get the information you want is to write two separate queries. This concept can be difficult to understand when you start using SQL for the first time after exclusively using Excel, but soon you will be come accustomed to it.

By the way, this diagram also shows you why some platforms and some queries in some platforms crash when you try to use aliases or derived fields in WHERE, GROUP BY, or HAVING clauses. If the SELECT statement hasn't been run yet, the alias or derived fields won't be available (as a reminder, some database systems--like MySQL--have found ways to overcome this issue). On the other hand, SELECT is executed before ORDER BY clauses. That means most database systems should be able to use aliases and derived fields in ORDER BY clauses.

Now that you are knowledgeable about the common pitfalls caused by GROUP BY, you are ready to perform one of the most powerful and fundamental utlities of a relational database: JOINS! Watch the next video to learn more about how joins work.

In [12]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid) AS Num_Dog, breed_type, AVG(weight) AS Avg_Weight
From dogs
GROUP BY breed_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[12]:

Num_Dog	breed_type	Avg_Weight
5568	Cross Breed	37.3509
9499	Mixed Breed/ Other/ I Don't Know	38.3682
1160	Popular Hybrid	36.8103
18823	Pure Breed	42.0353

In [13]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid) AS Num_Dog, breed_type, weight
From dogs
GROUP BY breed_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[13]:

Num_Dog	breed_type	weight
5568	Cross Breed	0
9499	Mixed Breed/ Other/ I Don't Know	50
1160	Popular Hybrid	70
18823	Pure Breed	50

In [14]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid) AS Num_Dog, breed_type
From dogs
GROUP BY breed_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[14]:

Num_Dog	breed_type
5568	Cross Breed
9499	Mixed Breed/ Other/ I Don't Know
1160	Popular Hybrid
18823	Pure Breed

In [17]:

```
%%sql
SELECT DISTINCT dog_guid AS Num_Dog, breed_type, AVG(weight) AS Avg_Weight, weight
From dogs
GROUP BY breed_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[17]:

Num_Dog	breed_type	Avg_Weight	weight
fd27c5be-7144-11e5-ba71-058fbc01cf0b	Cross Breed	37.3509	0
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	Mixed Breed/ Other/ I Don't Know	38.3682	50
fd27d248-7144-11e5-ba71-058fbc01cf0b	Popular Hybrid	36.8103	70
fd27b272-7144-11e5-ba71-058fbc01cf0b	Pure Breed	42.0353	50

In [19]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid) AS Num_Dog, breed_type, AVG(weight) AS Avg_Weight, weight
From dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[19]:

Num_Dog	breed_type	Avg_Weight	weight
35050	Pure Breed	40.1244	50

In [ ]:

```

```

MySQL Exercise 7: Joining Tables with Inner Joins

Before completing these exercises, I strongly recommend that you watch the video called "What are Joins?" that describe what joins are, and how different types of joins work.

As one of the last building blocks we need to address our Dognition analysis questions, in this lesson we will learn how to combine tables using inner joins.

1. Inner Joins between 2 tables

To begin, load the sql library, connect to the Dognition database, and set the Dognition database as the default.

In [1]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

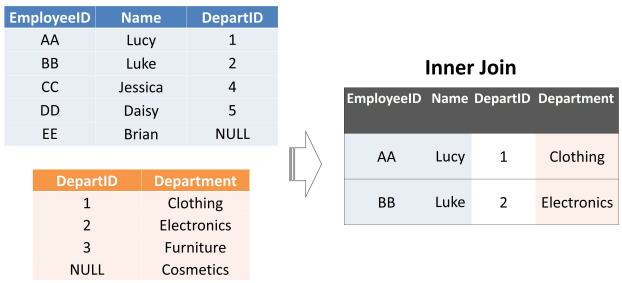
```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[1]:

```
[]
```

Recall that tables in relational databases are linked through primary keys and sometimes other fields that are common to multiple tables (as is the case with our Dognition data set). Our goal when we execute a JOIN or make a joined table is to use those common columns to let the database figure out which rows in one table match up to which rows in another table. Once that mapping is established using at least one common field or column, the database can pull any columns you want out of the mapped, or joined, tables and output the matched data to one common table.

An inner join is a join that outputs only rows that have an exact match in both tables being joined:



To illustrate how this works, let's find out whether dog owners that are particularly surprised by their dog's performance on Dognition tests tend to own similar breeds (or breed types, or breed groups) of dogs. There are many ways to address this question, but let's start by focusing on the dog owners who provided at least 10 ratings for one or more of their dogs in the ratings table. Of these owners, which 200 owners reported the highest average amount of surprise at their dog's performance, and what was the breed, breed_type, and breed_group of each of these owner's dog?

The surprise ratings are stored in the reviews table. The dog breed information is provided in the dogs table. There are two columns that are common to both tables: user_guid and dog_guid. How do we use the common columns to combine information from the two tables?

To join the tables, you can use a WHERE clause and add a couple of details to the FROM clause so that the database knows from what table each field in your SELECT clause comes.

First, start by adding all the columns we want to examine to the SELECT statement:

```
SELECT dog_guid AS DogID, user_guid AS UserID, AVG(rating) AS AvgRating,
       COUNT(rating) AS NumRatings, breed, breed_group, breed_type
```

then list all the tables from which the fields we are interested in come, separated by commas (with no comma at the end of the list):

```
FROM dogs, reviews
```

then add the other restrictions:

```
GROUP BY user_guid, dog_guid, breed, breed_group, breed_type
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 200
```

Try running this query and see what happens:

In [3]:

```
%%sql
SELECT dog_guid AS DogID, user_guid AS UserID, AVG(rating) AS AvgRating, COUNT(rating) AS NumRatings, breed, bree
d_group, breed_type
FROM dogs, reviews
GROUP BY user_guid, dog_guid, breed, breed_group, breed_type
HAVING NumRating >= 10
ORDER BY AvgRating DESC
LIMIT 200;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
(MySQLdb._exceptions.OperationalError) (1052, "Column 'dog_guid' in field list is ambiguous")
[SQL: SELECT dog_guid AS DogID, user_guid AS UserID, AVG(rating) AS AvgRating, COUNT(rating) AS NumR
atings, breed, breed_group, breed_type
FROM dogs, reviews
GROUP BY user_guid, dog_guid, breed, breed_group, breed_type
HAVING NumRating >= 10
ORDER BY AvgRating DESC
LIMIT 200;]
(Background on this error at: http://sqlalche.me/e/e3q8)
```

You should receive an error message stating that the identity of dog_guid and user_guid in the field list is ambiguous. The reason is that the column title exists in both tables, and MySQL doesn't know which one we want. We have to specify the table name before stating the field name, and separate the two names by a period (NOTE: read this entire section before deciding whether you want to execute this query)<mark>:

```
SELECT dogs.dog_guid AS DogID, dogs.user_guid AS UserID, AVG(reviews.rating) AS AvgRating,
       COUNT(reviews.rating) AS NumRatings, dogs.breed, dogs.breed_group, dogs.breed_type
FROM dogs, reviews
GROUP BY dogs.user_guid, dogs.dog_guid, dogs.breed, dogs.breed_group, dogs.breed_type
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 200
```

You can also take advantage of aliases so that you don't have to write out the name of the tables each time. Here I will introduce another syntax for aliases that omits the AS completely. In this syntax, the alias is whatever word (or phrase, if you use quotation marks) follows immediately after the field or table name, separated by a space. Some people find it easier to read queries if you use the syntax without an AS to create table aliases, but retain the syntax with an AS to create field aliases (but both syntaxes work for field aliases and table aliases). So we could write:

```
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating,
       COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM dogs d, reviews r
GROUP BY DogID, UserID, d.breed, d.breed_group, d.breed_type
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 200
```

I am tempted to tell you to run this query so that you will see what happens, but instead, I will explain what will happen and let you decide if you want to see what the output looks...and feels...like.

There is nothing built into the database table definitions that can instruct the server how to combine the tables on its own (remember, this is how relational databases save space and remain flexible). Further, the query as written does not tell the database how the two tables are related. As a consequence, rather than match up the two tables according to the values in the user_id and/or dog_id column, the database will do the only thing it knows how to do which is output every single combination of the records in the dogs table with the records in the reviews table. In other words, every single row of the dogs table will get paired with every single row of the reviews table. This is known as a Cartesian product. Not only will it be a heavy burden on the database to output a table that has the full length of one table multiplied times the full length of another (and frustrating to you, because the query would take a very long time to run), the output would be close to useless.

To prevent this from happening, tell the database how to relate the tables in the WHERE clause:

```
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating,
       COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM dogs d, reviews r
WHERE d.dog_guid=r.dog_guid
GROUP BY UserID, DogID, d.breed, d.breed_group, d.breed_type
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 200
```

To be very careful and exclude any incorrect dog_guid or user_guid entries, you can include both shared columns in the WHERE clause:

```
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating,
       COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM dogs d, reviews r
WHERE d.dog_guid=r.dog_guid AND d.user_guid=r.user_guid
GROUP BY UserID, DogID, d.breed, d.breed_group, d.breed_type
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 200
```

Try running this query now:

In [6]:

```
%%sql
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.b
reed, d.breed_group, d.breed_type
FROM dogs d, reviews r
WHERE d.dog_guid=r.dog_guid
GROUP BY d.user_guid, d.dog_guid, d.breed, d.breed_group, d.breed_type
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 200;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
38 rows affected.
```

Out[6]:

DogID	UserID	AvgRating	NumRatings	breed	breed_group	breed_type
fdbf39f8-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	8.0000	12	Canaan Dog	Herding	Pure Breed
fdc09a82-7144-11e5-ba71-058fbc01cf0b	ce99bb12-7144-11e5-ba71-058fbc01cf0b	5.0000	10	Golden Doodle	None	Popular Hybrid
fdbef330-7144-11e5-ba71-058fbc01cf0b	ce984d2c-7144-11e5-ba71-058fbc01cf0b	4.5385	13	Havanese	Toy	Pure Breed
fdc0518a-7144-11e5-ba71-058fbc01cf0b	ce99661c-7144-11e5-ba71-058fbc01cf0b	3.7333	15	Mixed	None	Mixed Breed/ Other/ I Don't Know
fd684cf6-7144-11e5-ba71-058fbc01cf0b	ce473856-7144-11e5-ba71-058fbc01cf0b	3.7222	18	Labrador Retriever	Sporting	Pure Breed
fdbf66e4-7144-11e5-ba71-058fbc01cf0b	ce98b9b0-7144-11e5-ba71-058fbc01cf0b	3.5000	14	Golden Retriever	Sporting	Pure Breed
fdc09b0e-7144-11e5-ba71-058fbc01cf0b	ce99bb76-7144-11e5-ba71-058fbc01cf0b	3.2308	13	Golden Doodle	None	Popular Hybrid
fdbeedc2-7144-11e5-ba71-058fbc01cf0b	ce9847dc-7144-11e5-ba71-058fbc01cf0b	3.1538	13	Eurasier	None	Pure Breed
fdc180be-7144-11e5-ba71-058fbc01cf0b	ce9a7110-7144-11e5-ba71-058fbc01cf0b	3.0625	16	Chihuahua-Dachshund Mix	None	Cross Breed
fd6aabd6-7144-11e5-ba71-058fbc01cf0b	ce6cd804-7144-11e5-ba71-058fbc01cf0b	3.0000	13	American Pit Bull Terrier	None	Pure Breed
fdbfc0b2-7144-11e5-ba71-058fbc01cf0b	ce98f3bc-7144-11e5-ba71-058fbc01cf0b	2.6923	13	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbedef4-7144-11e5-ba71-058fbc01cf0b	ce982e46-7144-11e5-ba71-058fbc01cf0b	2.4615	13	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbec8ce-7144-11e5-ba71-058fbc01cf0b	ce980e3e-7144-11e5-ba71-058fbc01cf0b	2.4545	11	Basset Hound	Hound	Pure Breed
fdbfe22c-7144-11e5-ba71-058fbc01cf0b	ce99049c-7144-11e5-ba71-058fbc01cf0b	2.4000	10	Irish Water Spaniel	Sporting	Pure Breed
fdbeff06-7144-11e5-ba71-058fbc01cf0b	ce9856f0-7144-11e5-ba71-058fbc01cf0b	2.4000	10	Bearded Collie	Herding	Pure Breed
fdbed7ce-7144-11e5-ba71-058fbc01cf0b	ce981dc0-7144-11e5-ba71-058fbc01cf0b	2.3333	18	Havanese	Toy	Pure Breed
fdc1c31c-7144-11e5-ba71-058fbc01cf0b	ce9ac674-7144-11e5-ba71-058fbc01cf0b	2.3077	13	Labrador Retriever	Sporting	Pure Breed
fdc0c49e-7144-11e5-ba71-058fbc01cf0b	ce99d674-7144-11e5-ba71-058fbc01cf0b	2.2778	18	Other	None	Mixed Breed/ Other/ I Don't Know
fdc19b30-7144-11e5-ba71-058fbc01cf0b	ce9aab30-7144-11e5-ba71-058fbc01cf0b	2.2222	18	Brittany	Sporting	Pure Breed
fdbedf94-7144-11e5-ba71-058fbc01cf0b	ce982ed2-7144-11e5-ba71-058fbc01cf0b	2.0000	11	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc0c502-7144-11e5-ba71-058fbc01cf0b	ce99d732-7144-11e5-ba71-058fbc01cf0b	2.0000	15	Golden Retriever	Sporting	Pure Breed
fdbfddfe-7144-11e5-ba71-058fbc01cf0b	ce99049c-7144-11e5-ba71-058fbc01cf0b	1.9375	16	Irish Water Spaniel	Sporting	Pure Breed
fd7104ae-7144-11e5-ba71-058fbc01cf0b	ce7212a6-7144-11e5-ba71-058fbc01cf0b	1.9000	10	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc198d8-7144-11e5-ba71-058fbc01cf0b	ce9a9500-7144-11e5-ba71-058fbc01cf0b	1.8000	10	German Shepherd Dog	Herding	Pure Breed
fdbe923c-7144-11e5-ba71-058fbc01cf0b	ce97f926-7144-11e5-ba71-058fbc01cf0b	1.7647	17	Beagle	Hound	Pure Breed
fd6a7490-7144-11e5-ba71-058fbc01cf0b	ce6cf3fc-7144-11e5-ba71-058fbc01cf0b	1.7500	20	Golden Retriever	Sporting	Pure Breed
fdbef5e2-7144-11e5-ba71-058fbc01cf0b	ce984f70-7144-11e5-ba71-058fbc01cf0b	1.7000	10	Havanese	Toy	Pure Breed
fdbf9f10-7144-11e5-ba71-058fbc01cf0b	ce98d6de-7144-11e5-ba71-058fbc01cf0b	1.6667	18	Australian Cattle Dog-Border Collie Mix	None	Cross Breed
fdbea952-7144-11e5-ba71-058fbc01cf0b	ce9805ce-7144-11e5-ba71-058fbc01cf0b	1.6667	15	Golden Doodle	None	Popular Hybrid
fdbf686a-7144-11e5-ba71-058fbc01cf0b	ce98bca8-7144-11e5-ba71-058fbc01cf0b	1.6000	10	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbf6144-7144-11e5-ba71-058fbc01cf0b	ce98b776-7144-11e5-ba71-058fbc01cf0b	1.5455	11	American Pit Bull Terrier-Labrador Retriever Mix	None	Cross Breed
fdc1edec-7144-11e5-ba71-058fbc01cf0b	ce9ae4b0-7144-11e5-ba71-058fbc01cf0b	1.5333	15	Poodle	Non-Sporting	Pure Breed
fdbe7cac-7144-11e5-ba71-	ce97e10c-7144-11e5-ba71-					

fdbe7cac-7144-11e5-ba71-058fbc01cf0b	ce97e10c-7144-11e5-ba71-058fbc01cf0b	1.3333	12	Labrador Retriever	Sporting	Pure Breed
fdbe8a9e-7144-11e5-ba71-058fbc01cf0b	ce97f1f6-7144-11e5-ba71-058fbc01cf0b	1.2500	16	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbfe56a-7144-11e5-ba71-058fbc01cf0b	ce99049c-7144-11e5-ba71-058fbc01cf0b	1.2143	14	Irish Water Spaniel	Sporting	Pure Breed
fdbf0456-7144-11e5-ba71-058fbc01cf0b	ce985d9e-7144-11e5-ba71-058fbc01cf0b	0.7500	16	English Springer Spaniel	Sporting	Pure Breed
fdc23ba8-7144-11e5-ba71-058fbc01cf0b	ce9b306e-7144-11e5-ba71-058fbc01cf0b	0.4211	19	Cockapoo	None	Popular Hybrid
fdbec040-7144-11e5-ba71-058fbc01cf0b	ce98098e-7144-11e5-ba71-058fbc01cf0b	0.4000	10	Mixed	None	Mixed Breed/ Other/ I Don't Know

In [5]:

```
%%sql
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.b
reed, d.breed_group, d.breed_type
FROM dogs d, reviews r
WHERE d.dog_guid=r.dog_guid AND d.user_guid=r.user_guid
GROUP BY d.user_guid, d.dog_guid, d.breed, d.breed_group, d.breed_type
HAVING NumRatings >= 10
ORDER BY AvgRating DESC
LIMIT 200;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
38 rows affected.
```

Out[5]:

DogID	UserID	AvgRating	NumRatings	breed	breed_group	breed_type
fdbf39f8-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	8.0000	12	Canaan Dog	Herding	Pure Breed
fdc09a82-7144-11e5-ba71-058fbc01cf0b	ce99bb12-7144-11e5-ba71-058fbc01cf0b	5.0000	10	Golden Doodle	None	Popular Hybrid
fdbef330-7144-11e5-ba71-058fbc01cf0b	ce984d2c-7144-11e5-ba71-058fbc01cf0b	4.5385	13	Havanese	Toy	Pure Breed
fdc0518a-7144-11e5-ba71-058fbc01cf0b	ce99661c-7144-11e5-ba71-058fbc01cf0b	3.7333	15	Mixed	None	Mixed Breed/ Other/ I Don't Know
fd684cf6-7144-11e5-ba71-058fbc01cf0b	ce473856-7144-11e5-ba71-058fbc01cf0b	3.7222	18	Labrador Retriever	Sporting	Pure Breed
fdbf66e4-7144-11e5-ba71-058fbc01cf0b	ce98b9b0-7144-11e5-ba71-058fbc01cf0b	3.5000	14	Golden Retriever	Sporting	Pure Breed
fdc09b0e-7144-11e5-ba71-058fbc01cf0b	ce99bb76-7144-11e5-ba71-058fbc01cf0b	3.2308	13	Golden Doodle	None	Popular Hybrid
fdbeedc2-7144-11e5-ba71-058fbc01cf0b	ce9847dc-7144-11e5-ba71-058fbc01cf0b	3.1538	13	Eurasier	None	Pure Breed
fdc180be-7144-11e5-ba71-058fbc01cf0b	ce9a7110-7144-11e5-ba71-058fbc01cf0b	3.0625	16	Chihuahua-Dachshund Mix	None	Cross Breed
fd6aabd6-7144-11e5-ba71-058fbc01cf0b	ce6cd804-7144-11e5-ba71-058fbc01cf0b	3.0000	13	American Pit Bull Terrier	None	Pure Breed
fdbfc0b2-7144-11e5-ba71-058fbc01cf0b	ce98f3bc-7144-11e5-ba71-058fbc01cf0b	2.6923	13	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbedef4-7144-11e5-ba71-058fbc01cf0b	ce982e46-7144-11e5-ba71-058fbc01cf0b	2.4615	13	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbec8ce-7144-11e5-ba71-058fbc01cf0b	ce980e3e-7144-11e5-ba71-058fbc01cf0b	2.4545	11	Basset Hound	Hound	Pure Breed
fdbfe22c-7144-11e5-ba71-058fbc01cf0b	ce99049c-7144-11e5-ba71-058fbc01cf0b	2.4000	10	Irish Water Spaniel	Sporting	Pure Breed
fdbeff06-7144-11e5-ba71-058fbc01cf0b	ce9856f0-7144-11e5-ba71-058fbc01cf0b	2.4000	10	Bearded Collie	Herding	Pure Breed

fdbed7ce-7144-11e5-ba71-058fbc01cf0b	ce981dc0-7144-11e5-ba71-058fbc01cf0b	2.3333	18	Havanese	Toy	Pure Breed
fdc1c31c-7144-11e5-ba71-058fbc01cf0b	ce9ac674-7144-11e5-ba71-058fbc01cf0b	2.3077	13	Labrador Retriever	Sporting	Pure Breed
fdc0c49e-7144-11e5-ba71-058fbc01cf0b	ce99d674-7144-11e5-ba71-058fbc01cf0b	2.2778	18	Other	None	Mixed Breed/ Other/ I Don't Know
fdc19b30-7144-11e5-ba71-058fbc01cf0b	ce9aab30-7144-11e5-ba71-058fbc01cf0b	2.2222	18	Brittany	Sporting	Pure Breed
fdbedf94-7144-11e5-ba71-058fbc01cf0b	ce982ed2-7144-11e5-ba71-058fbc01cf0b	2.0000	11	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc0c502-7144-11e5-ba71-058fbc01cf0b	ce99d732-7144-11e5-ba71-058fbc01cf0b	2.0000	15	Golden Retriever	Sporting	Pure Breed
fdbfddfe-7144-11e5-ba71-058fbc01cf0b	ce99049c-7144-11e5-ba71-058fbc01cf0b	1.9375	16	Irish Water Spaniel	Sporting	Pure Breed
fd7104ae-7144-11e5-ba71-058fbc01cf0b	ce7212a6-7144-11e5-ba71-058fbc01cf0b	1.9000	10	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc198d8-7144-11e5-ba71-058fbc01cf0b	ce9a9500-7144-11e5-ba71-058fbc01cf0b	1.8000	10	German Shepherd Dog	Herding	Pure Breed
fdbe923c-7144-11e5-ba71-058fbc01cf0b	ce97f926-7144-11e5-ba71-058fbc01cf0b	1.7647	17	Beagle	Hound	Pure Breed
fd6a7490-7144-11e5-ba71-058fbc01cf0b	ce6cf3fc-7144-11e5-ba71-058fbc01cf0b	1.7500	20	Golden Retriever	Sporting	Pure Breed
fdbef5e2-7144-11e5-ba71-058fbc01cf0b	ce984f70-7144-11e5-ba71-058fbc01cf0b	1.7000	10	Havanese	Toy	Pure Breed
fdbf9f10-7144-11e5-ba71-058fbc01cf0b	ce98d6de-7144-11e5-ba71-058fbc01cf0b	1.6667	18	Australian Cattle Dog-Border Collie Mix	None	Cross Breed
fdbea952-7144-11e5-ba71-058fbc01cf0b	ce9805ce-7144-11e5-ba71-058fbc01cf0b	1.6667	15	Golden Doodle	None	Popular Hybrid
fdbf686a-7144-11e5-ba71-058fbc01cf0b	ce98bca8-7144-11e5-ba71-058fbc01cf0b	1.6000	10	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbf6144-7144-11e5-ba71-058fbc01cf0b	ce98b776-7144-11e5-ba71-058fbc01cf0b	1.5455	11	American Pit Bull Terrier-Labrador Retriever Mix	None	Cross Breed
fdc1edec-7144-11e5-ba71-058fbc01cf0b	ce9ae4b0-7144-11e5-ba71-058fbc01cf0b	1.5333	15	Poodle	Non-Sporting	Pure Breed
fdbe7cac-7144-11e5-ba71-058fbc01cf0b	ce97e10c-7144-11e5-ba71-058fbc01cf0b	1.3333	12	Labrador Retriever	Sporting	Pure Breed
fdbe8a9e-7144-11e5-ba71-058fbc01cf0b	ce97f1f6-7144-11e5-ba71-058fbc01cf0b	1.2500	16	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbfe56a-7144-11e5-ba71-058fbc01cf0b	ce99049c-7144-11e5-ba71-058fbc01cf0b	1.2143	14	Irish Water Spaniel	Sporting	Pure Breed
fdbf0456-7144-11e5-ba71-058fbc01cf0b	ce985d9e-7144-11e5-ba71-058fbc01cf0b	0.7500	16	English Springer Spaniel	Sporting	Pure Breed
fdc23ba8-7144-11e5-ba71-058fbc01cf0b	ce9b306e-7144-11e5-ba71-058fbc01cf0b	0.4211	19	Cockapoo	None	Popular Hybrid
fdbec040-7144-11e5-ba71-058fbc01cf0b	ce98098e-7144-11e5-ba71-058fbc01cf0b	0.4000	10	Mixed	None	Mixed Breed/ Other/ I Don't Know

The query should execute quickly. This would NOT have been the case if you did not include the WHERE clause to combine the two tables. If you accidentally request a Cartesian product from datasets with billions of rows, you could be waiting for your query output for days (and will probably get in trouble with your database administrator). So always remember to tell the database how to join your tables!

Let's examine our joined table a bit further. The joined table outputted by the query above should have 38 rows, despite the fact that we set our LIMIT at 200. The reason for this is that it turns out that a relatively small number of customers provided 10 or more reviews. If you remove the HAVING and LIMIT BY clause from the query, you should end up with 389 rows. **Go ahead and try it:**

In [40]:

```
%%sql
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.b
reed, d.breed_group, d.breed_type
FROM dogs d, reviews r
WHERE d.dog_guid=r.dog_guid AND d.user_guid=r.user_guid
GROUP BY d.user_guid, d.dog_guid, d.breed, d.breed_group, d.breed_type
ORDER BY AvgRating DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[40]:

DogID	UserID	AvgRating	NumRatings	breed	breed_group	breed_type
fdc121dc-7144-11e5-ba71-058fbc01cf0b	ce9a310a-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbf94de-7144-11e5-ba71-058fbc01cf0b	ce98cc34-7144-11e5-ba71-058fbc01cf0b	9.0000	2	Pembroke Welsh Corgi	Herding	Pure Breed
fdc059fa-7144-11e5-ba71-058fbc01cf0b	ce996dc4-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Irish Water Spaniel	Sporting	Pure Breed
fdc019c2-7144-11e5-ba71-058fbc01cf0b	ce99489e-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbf178e-7144-11e5-ba71-058fbc01cf0b	ce986546-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Airedale Terrier	Terrier	Pure Breed
fdc19554-7144-11e5-ba71-058fbc01cf0b	ce9a8a24-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Miniature Pinscher	Toy	Pure Breed
fdc057ac-7144-11e5-ba71-058fbc01cf0b	ce996b9e-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbedd14-7144-11e5-ba71-058fbc01cf0b	ce975994-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc147ac-7144-11e5-ba71-058fbc01cf0b	ce9a545a-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Tibetan Terrier	Non-Sporting	Pure Breed
fdc1c880-7144-11e5-ba71-058fbc01cf0b	ce96c768-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Maltese	Toy	Pure Breed
fd41bca8-7144-11e5-ba71-058fbc01cf0b	ce7b055a-7144-11e5-ba71-058fbc01cf0b	8.0000	1	Labrador Retriever	Sporting	Pure Breed
fdbf39f8-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	8.0000	12	Canaan Dog	Herding	Pure Breed
fdc1e1a8-7144-11e5-ba71-058fbc01cf0b	ce9ad934-7144-11e5-ba71-058fbc01cf0b	7.5000	2	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc2417a-7144-11e5-ba71-058fbc01cf0b	ce9b35e6-7144-11e5-ba71-058fbc01cf0b	7.0000	1	Bichon Frise	Non-Sporting	Pure Breed
fdbe7a40-7144-11e5-ba71-058fbc01cf0b	ce97df9a-7144-11e5-ba71-058fbc01cf0b	7.0000	6	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbf940c-7144-11e5-ba71-058fbc01cf0b	ce98cb80-7144-11e5-ba71-058fbc01cf0b	7.0000	1	Soft Coated Wheaten Terrier	Terrier	Pure Breed
fdc1baca-7144-11e5-ba71-058fbc01cf0b	ce9ac2fa-7144-11e5-ba71-058fbc01cf0b	7.0000	1	Poodle-Miniature Schnauzer Mix	None	Cross Breed
fdbe8774-7144-11e5-ba71-058fbc01cf0b	ce97ebfc-7144-11e5-ba71-058fbc01cf0b	7.0000	4	Labrador Retriever	Sporting	Pure Breed
fdc1706a-7144-11e5-ba71-058fbc01cf0b	ce9a6846-7144-11e5-ba71-058fbc01cf0b	6.8000	5	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc0c2aa-7144-11e5-ba71-058fbc01cf0b	ce99d322-7144-11e5-ba71-058fbc01cf0b	6.5000	2	Border Collie	Herding	Pure Breed

In [ ]:

```
CODE

%%sql
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.b
reed, d.breed_group, d.breed_type
FROM dogs d, reviews r
WHERE d.dog_guid=r.dog_guid AND d.user_guid=r.user_guid
GROUP BY d.user_guid, d.dog_guid, d.breed, d.breed_group, d.breed_type
ORDER BY AvgRating DESC;

OUTPUT

* mysql://studentuser:***@localhost/dognitiondb
395 rows affected.
```

It's clear from looking at this output that (A) not many customers provided ratings, and (B) when they did, they usually were not very surprised by their dog's performance. Therefore, these ratings are probably not going to provide a lot of instructive insight into how to improve Dognition's completion rate. However, the ratings table still provides a great opportunity to illustrate the results of different types of joins.

To help prepare us for this:

Questions 1-4: How many unique dog_guids and user_guids are there in the reviews and dogs table independently?

In [14]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM dogs
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[14]:

COUNT(DISTINCT dog_guid)
35050

In [20]:

```
%%sql
SELECT COUNT(DISTINCT user_guid)
FROM dogs;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[20]:

COUNT(DISTINCT user_guid)
30967

In [21]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM reviews;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[21]:

COUNT(DISTINCT dog_guid)
5991

In [22]:

```
%%sql
SELECT COUNT(DISTINCT user_guid)
FROM reviews;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[22]:

COUNT(DISTINCT user_guid)
5586

These counts indicate some important things:

- Many customers in both the reviews and the dogs table have multiple dogs
- There are many more unique dog_guids and user_guids in the dogs table than the reviews table
- There are many more unique dog_guids and user_guids in the reviews table than in the output of our inner join

Let's test one more thing.

Try the inner join query once with just the dog_guid or once with just the user_guid clause in the WHERE statement:

In [38]:

```
%%sql
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.b
reed, d.breed_group, d.breed_type
FROM dogs d, reviews r
WHERE d.dog_guid=r.dog_guid
GROUP BY d.user_guid, d.dog_guid, d.breed, d.breed_group, d.breed_type
ORDER BY AvgRating DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[38]:

DogID	UserID	AvgRating	NumRatings	breed	breed_group	breed_type
fdc121dc-7144-11e5-ba71-058fbc01cf0b	ce9a310a-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbf94de-7144-11e5-ba71-058fbc01cf0b	ce98cc34-7144-11e5-ba71-058fbc01cf0b	9.0000	2	Pembroke Welsh Corgi	Herding	Pure Breed
fdc059fa-7144-11e5-ba71-058fbc01cf0b	ce996dc4-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Irish Water Spaniel	Sporting	Pure Breed
fdc019c2-7144-11e5-ba71-058fbc01cf0b	ce99489e-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbf178e-7144-11e5-ba71-058fbc01cf0b	ce986546-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Airedale Terrier	Terrier	Pure Breed
fdc19554-7144-11e5-ba71-058fbc01cf0b	ce9a8a24-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Miniature Pinscher	Toy	Pure Breed
fdc057ac-7144-11e5-ba71-058fbc01cf0b	ce996b9e-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbedd14-7144-11e5-ba71-058fbc01cf0b	ce975994-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc147ac-7144-11e5-ba71-058fbc01cf0b	ce9a545a-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Tibetan Terrier	Non-Sporting	Pure Breed
fdc1c880-7144-11e5-ba71-058fbc01cf0b	ce96c768-7144-11e5-ba71-058fbc01cf0b	9.0000	1	Maltese	Toy	Pure Breed
fd41bca8-7144-11e5-ba71-058fbc01cf0b	ce7b055a-7144-11e5-ba71-058fbc01cf0b	8.0000	1	Labrador Retriever	Sporting	Pure Breed
fdbf39f8-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	8.0000	12	Canaan Dog	Herding	Pure Breed
fdc1e1a8-7144-11e5-ba71-058fbc01cf0b	ce9ad934-7144-11e5-ba71-058fbc01cf0b	7.5000	2	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc2417a-7144-11e5-ba71-058fbc01cf0b	ce9b35e6-7144-11e5-ba71-058fbc01cf0b	7.0000	1	Bichon Frise	Non-Sporting	Pure Breed
fdbe7a40-7144-11e5-ba71-058fbc01cf0b	ce97df9a-7144-11e5-ba71-058fbc01cf0b	7.0000	6	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdbf940c-7144-11e5-ba71-058fbc01cf0b	ce98cb80-7144-11e5-ba71-058fbc01cf0b	7.0000	1	Soft Coated Wheaten Terrier	Terrier	Pure Breed
fdc1baca-7144-11e5-ba71-058fbc01cf0b	ce9ac2fa-7144-11e5-ba71-058fbc01cf0b	7.0000	1	Poodle-Miniature Schnauzer Mix	None	Cross Breed
fdbe8774-7144-11e5-ba71-058fbc01cf0b	ce97ebfc-7144-11e5-ba71-058fbc01cf0b	7.0000	4	Labrador Retriever	Sporting	Pure Breed
fdc1706a-7144-11e5-ba71-058fbc01cf0b	ce9a6846-7144-11e5-ba71-058fbc01cf0b	6.8000	5	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdc0c2aa-7144-11e5-ba71-058fbc01cf0b	ce99d322-7144-11e5-ba71-058fbc01cf0b	6.5000	2	Border Collie	Herding	Pure Breed

CODE

%%sql SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type FROM dogs d, reviews r WHERE d.dog_guid=r.dog_guid GROUP BY d.user_guid, d.dog_guid, d.breed, d.breed_group, d.breed_type ORDER BY AvgRating DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 395 rows affected.

In [36]:

```
%%sql
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.b
reed, d.breed_group, d.breed_type
FROM dogs d, reviews r
WHERE d.user_guid=r.user_guid
GROUP BY d.user_guid, d.dog_guid, d.breed, d.breed_group, d.breed_type
ORDER BY NumRatings DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[36]:

DogID	UserID	AvgRating	NumRatings	breed	breed_group	breed_type
fd742db4-7144-11e5-ba71-058fbc01cf0b	ce7498be-7144-11e5-ba71-058fbc01cf0b	2.7011	87	Mixed	None	Mixed Breed/ Other/ I Don't Know
fd74e452-7144-11e5-ba71-058fbc01cf0b	ce7498be-7144-11e5-ba71-058fbc01cf0b	2.7011	87	Labrador Retriever-Poodle Mix		Cross Breed
fd74e4c0-7144-11e5-ba71-058fbc01cf0b	ce7498be-7144-11e5-ba71-058fbc01cf0b	2.7011	87	Poodle-Labrador Retriever Mix	None	Cross Breed
fd754abe-7144-11e5-ba71-058fbc01cf0b	ce7498be-7144-11e5-ba71-058fbc01cf0b	2.7011	87	Chihuahua	Toy	Pure Breed
fd760f80-7144-11e5-ba71-058fbc01cf0b	ce765ffa-7144-11e5-ba71-058fbc01cf0b	3.7500	84	German Shepherd Dog	Herding	Pure Breed
fd760fee-7144-11e5-ba71-058fbc01cf0b	ce765ffa-7144-11e5-ba71-058fbc01cf0b	3.7500	84	German Shepherd Dog	Herding	Pure Breed
fd7b085a-7144-11e5-ba71-058fbc01cf0b	ce765ffa-7144-11e5-ba71-058fbc01cf0b	3.7500	84	German Shepherd Dog	Herding	Pure Breed
fd7b08b4-7144-11e5-ba71-058fbc01cf0b	ce765ffa-7144-11e5-ba71-058fbc01cf0b	3.7500	84	German Shepherd Dog	Herding	Pure Breed
fd694bb0-7144-11e5-ba71-058fbc01cf0b	ce66ea52-7144-11e5-ba71-058fbc01cf0b	3.1781	73	Weimaraner	Sporting	Pure Breed
fd694ee4-7144-11e5-ba71-058fbc01cf0b	ce66ea52-7144-11e5-ba71-058fbc01cf0b	3.1781	73	Weimaraner	Sporting	Pure Breed
fd694f48-7144-11e5-ba71-058fbc01cf0b	ce66ea52-7144-11e5-ba71-058fbc01cf0b	3.1781	73	Weimaraner	Sporting	Pure Breed
fd771a38-7144-11e5-ba71-058fbc01cf0b	ce77539c-7144-11e5-ba71-058fbc01cf0b	2.4068	59	Rhodesian Ridgeback-American Staffordshire Terrier Mix	None	Cross Breed
fd771ace-7144-11e5-ba71-058fbc01cf0b	ce77539c-7144-11e5-ba71-058fbc01cf0b	2.4068	59	Brittany-Basset Hound Mix	None	Cross Breed
fd727514-7144-11e5-ba71-058fbc01cf0b	ce734284-7144-11e5-ba71-058fbc01cf0b	2.7115	52	I Don't Know	None	Mixed Breed/ Other/ I Don't Know
fd7416c6-7144-11e5-ba71-058fbc01cf0b	ce734284-7144-11e5-ba71-058fbc01cf0b	2.7115	52	Mixed	None	Mixed Breed/ Other/ I Don't Know
fd741734-7144-11e5-ba71-058fbc01cf0b	ce734284-7144-11e5-ba71-058fbc01cf0b	2.7115	52	American Pit Bull Terrier-Labrador Retriever Mix	None	Cross Breed
fd7654e0-7144-11e5-ba71-058fbc01cf0b	ce76852a-7144-11e5-ba71-058fbc01cf0b	4.3000	50	Border Collie	Herding	Pure Breed
fd765544-7144-11e5-ba71-058fbc01cf0b	ce76852a-7144-11e5-ba71-058fbc01cf0b	4.3000	50	Border Collie	Herding	Pure Breed
fdb1d66e-7144-11e5-ba71-058fbc01cf0b	ce8866e6-7144-11e5-ba71-058fbc01cf0b	4.7143	49	Mixed	None	Mixed Breed/ Other/ I Don't Know
fdb470fe-7144-11e5-ba71-058fbc01cf0b	ce8866e6-7144-11e5-ba71-058fbc01cf0b	4.7143	49	Mixed	None	Mixed Breed/ Other/ I Don't Know

CODE

%%sql SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type FROM dogs d, reviews r WHERE d.user_guid=r.user_guid GROUP BY d.user_guid, d.dog_guid, d.breed, d.breed_group, d.breed_type ORDER BY NumRatings DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 6267 rows affected.

When you run the query by joining on the dog_guid only, you still get 389 rows in your output. When you run the query by joining on the user_guid only, you get 5586 rows in your output. This means that:

- All of the user_guids in the reviews table are in the dogs table
- Only 389 of the over 5000 dog_guids in the reviews table are in the dogs table

Perhaps most importantly for our current purposes, these COUNT queries show you that *inner joins only output the data from rows that have equivalent values in both tables being joined*. If you wanted to include all the dog_guids or user_guids in one or both of the tables, you would have to use an outer join, which we will practice in the next lesson.

Try an inner join on your own.

Question 5: How would you extract the user_guid, dog_guid, breed, breed_type, and breed_group for all animals who completed the "Yawn Warm-up" game (you should get 20,845 rows if you join on dog_guid only)?

In [22]:

```
%%sql
SELECT d.user_guid AS UserID, d.dog_guid AS DogID, d.breed, d.breed_type, d.breed_group
FROM dogs d, complete_tests c
WHERE d.dog_guid=c.dog_guid AND test_name='Yawn Warm-up'
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[22]:

UserID	DogID	breed	breed_type	breed_group
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	Pure Breed	Toy
ce135f2c-7144-11e5-ba71-058fbc01cf0b	fd27bbbe-7144-11e5-ba71-058fbc01cf0b	Mixed	Mixed Breed/ Other/ I Don't Know	None
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	Pure Breed	Toy
ce136ac6-7144-11e5-ba71-058fbc01cf0b	fd27c5be-7144-11e5-ba71-058fbc01cf0b	Shih Tzu-Poodle Mix	Cross Breed	None
ce13615c-7144-11e5-ba71-058fbc01cf0b	fd27b948-7144-11e5-ba71-058fbc01cf0b	Siberian Husky	Pure Breed	Working
ce1353d8-7144-11e5-ba71-058fbc01cf0b	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog	Pure Breed	Herding
ce136ee0-7144-11e5-ba71-058fbc01cf0b	fd27c852-7144-11e5-ba71-058fbc01cf0b	Pug	Pure Breed	Toy
ce136f94-7144-11e5-ba71-058fbc01cf0b	fd27c8d4-7144-11e5-ba71-058fbc01cf0b	Boxer	Pure Breed	Working
ce13750c-7144-11e5-ba71-058fbc01cf0b	fd27cf28-7144-11e5-ba71-058fbc01cf0b	Chesapeake Bay Retriever	Pure Breed	Sporting
ce136e36-7144-11e5-ba71-058fbc01cf0b	fd27c7d0-7144-11e5-ba71-058fbc01cf0b	Vizsla	Pure Breed	Sporting
ce1373ae-7144-11e5-ba71-058fbc01cf0b	fd27cea6-7144-11e5-ba71-058fbc01cf0b	Mixed	Mixed Breed/ Other/ I Don't Know	None
ce136c24-7144-11e5-ba71-058fbc01cf0b	fd27c74e-7144-11e5-ba71-058fbc01cf0b	German Shepherd Dog-Pembroke Welsh Corgi Mix	Cross Breed	None
ce1377b4-7144-11e5-ba71-058fbc01cf0b	fd27d02c-7144-11e5-ba71-058fbc01cf0b	Belgian Malinois	Pure Breed	Herding
ce138312-7144-11e5-ba71-058fbc01cf0b	fd27dd38-7144-11e5-ba71-058fbc01cf0b	Parson Russell Terrier-Beagle Mix	Cross Breed	None
ce13807e-7144-11e5-ba71-058fbc01cf0b	fd27db08-7144-11e5-ba71-058fbc01cf0b	German Shepherd Dog	Pure Breed	Herding
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ed46-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	Pure Breed	Toy
ce135194-7144-11e5-ba71-058fbc01cf0b	fd27e026-7144-11e5-ba71-058fbc01cf0b	Dalmatian	Pure Breed	Non-Sporting
ce135194-7144-11e5-ba71-058fbc01cf0b	fd27e0d0-7144-11e5-ba71-058fbc01cf0b	I Don't Know	Mixed Breed/ Other/ I Don't Know	None
ce138722-7144-11e5-ba71-058fbc01cf0b	fd27eae4-7144-11e5-ba71-058fbc01cf0b	Australian Terrier	Pure Breed	Terrier
ce1375b6-7144-11e5-ba71-058fbc01cf0b	fd27cfaa-7144-11e5-ba71-058fbc01cf0b	Border Collie	Pure Breed	Herding

CODE

%%sql SELECT d.user_guid AS UserID, d.dog_guid AS DogID, d.breed, d.breed_type, d.breed_group FROM dogs d, complete_tests c WHERE d.dog_guid=c.dog_guid AND test_name='Yawn Warm-up'

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 20845 rows affected.

2. Joining More than 2 Tables

In theory, you can join as many tables together as you want or need. To join multiple tables you take the same approach as we took when we were joining two tables together: list all the fields you want to extract in the SELECT statement, specify which table they came from in the SELECT statement, list all the tables from which you will need to extract the fields in the FROM statement, and then tell the database how to connect the tables in the WHERE statement.

To extract the user_guid, user's state of residence, user's zip code, dog_guid, breed, breed_type, and breed_group for all animals who completed the "Yawn Warm-up" game, you might be tempted to query:

```
SELECT c.user_guid AS UserID, u.state, u.zip, d.dog_guid AS DogID, d.breed, d.breed_type, d.breed_group
FROM dogs d, complete_tests c, users u
WHERE d.dog_guid=c.dog_guid
   AND c.user_guid=u.user_guid
   AND c.test_name="Yawn Warm-up";
```

This query focuses the relationships primarily on the complete_tests table. However, it turns out that our Dognition dataset has only NULL values in the user_guid column of the complete_tests table. If you were to execute the query above, you would not get an error message, but your output would have 0 rows. However, the power of relational databases will come in handy here. You can use the dogs table to link the complete_tests and users table (pay attention to the difference between the WHERE statement in this query vs. the WHERE statement in the query above):

```
SELECT d.user_guid AS UserID, u.state, u.zip, d.dog_guid AS DogID, d.breed, d.breed_type, d.breed_group
FROM dogs d, complete_tests c, users u
WHERE d.dog_guid=c.dog_guid
   AND d.user_guid=u.user_guid
   AND c.test_name="Yawn Warm-up";
```

Of note, joins are very resource intensive, so try not to join unnecessarily. In general, the more joins you have to execute, the slower your query performance will be.

Question 6: How would you extract the user_guid, membership_type, and dog_guid of all the golden retrievers who completed at least 1 Dognition test (you should get 711 rows)?

In [25]:

```
%%sql
SELECT d.user_guid AS UserID, d.dog_guid AS DogID, u.state, u.zip, d.breed,d.breed_type,d.breed_group
FROM dogs d, users u, complete_tests c
WHERE d.dog_guid=c.dog_guid AND d.user_guid=u.user_guid AND c.test_name="Yawn Warm-up"
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[25]:

UserID	DogID	state	zip	breed	breed_type	breed_group
ce134e42-7144-11e5-ba71-058fbc01cf0b	fd27b272-7144-11e5-ba71-058fbc01cf0b	ND	58201	Labrador Retriever	Pure Breed	Sporting
ce134e42-7144-11e5-ba71-058fbc01cf0b	fd417cac-7144-11e5-ba71-058fbc01cf0b	ND	58201	Mixed	Mixed Breed/ Other/ I Don't Know	None
ce1353d8-7144-11e5-ba71-058fbc01cf0b	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	MA	1005	Shetland Sheepdog	Pure Breed	Herding
ce1353d8-7144-11e5-ba71-058fbc01cf0b	fd3fb0f2-7144-11e5-ba71-058fbc01cf0b	MA	1005	Shetland Sheepdog	Pure Breed	Herding
ce135ab8-7144-11e5-ba71-058fbc01cf0b	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	CT	6820	Golden Retriever	Pure Breed	Sporting
ce13507c-7144-11e5-ba71-058fbc01cf0b	fd27b79a-7144-11e5-ba71-058fbc01cf0b	IL	60093	Golden Retriever	Pure Breed	Sporting
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27e9a4-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ed46-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd3cf718-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd3d587a-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd3fbfe8-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd41c400-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd42e33a-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd453b6c-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce13615c-7144-11e5-ba71-058fbc01cf0b	fd27b948-7144-11e5-ba71-058fbc01cf0b	WA	98001	Siberian Husky	Pure Breed	Working
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy
ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27e9a4-7144-11e5-ba71-058fbc01cf0b	NC	27606	Shih Tzu	Pure Breed	Toy

CODE

%%sql SELECT d.user_guid AS UserID, d.dog_guid AS DogID, u.state, u.zip, d.breed,d.breed_type,d.breed_group FROM dogs d, users u, complete_tests c WHERE d.dog_guid=c.dog_guid AND d.user_guid=u.user_guid AND c.test_name="Yawn Warm-up";

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 25992 rows affected.

QUESTION 6 ANSWER

In [32]:

```
%%sql
SELECT DISTINCT d.user_guid AS UserID, u.membership_type, d.dog_guid AS DogID, d.breed
FROM dogs d, users u, complete_tests c
WHERE d.dog_guid=c.dog_guid AND d.user_guid=u.user_guid AND d.breed="Golden Retriever"
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[32]:

UserID	membership_type	DogID	breed
ce135ab8-7144-11e5-ba71-058fbc01cf0b	1	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce13507c-7144-11e5-ba71-058fbc01cf0b	1	fd27b79a-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce1389d4-7144-11e5-ba71-058fbc01cf0b	1	fd27efb2-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce21f122-7144-11e5-ba71-058fbc01cf0b	2	fd3d03fc-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce220bb2-7144-11e5-ba71-058fbc01cf0b	2	fd3d10cc-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce2237f4-7144-11e5-ba71-058fbc01cf0b	2	fd3d3b24-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce2243e8-7144-11e5-ba71-058fbc01cf0b	2	fd3d4b8c-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce240b24-7144-11e5-ba71-058fbc01cf0b	2	fd3fe96e-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce135cf2-7144-11e5-ba71-058fbc01cf0b	2	fd7af63a-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce24476a-7144-11e5-ba71-058fbc01cf0b	2	fd404ff8-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce245cdc-7144-11e5-ba71-058fbc01cf0b	2	fd4059e4-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce245cdc-7144-11e5-ba71-058fbc01cf0b	2	fd45eb98-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce246be6-7144-11e5-ba71-058fbc01cf0b	2	fd40738e-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce246be6-7144-11e5-ba71-058fbc01cf0b	2	fd45d6ee-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce249774-7144-11e5-ba71-058fbc01cf0b	1	fd408ed2-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce249c6a-7144-11e5-ba71-058fbc01cf0b	2	fd40945e-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce2258a6-7144-11e5-ba71-058fbc01cf0b	2	fd6487c4-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce24c1c2-7144-11e5-ba71-058fbc01cf0b	2	fd40ee72-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce24a106-7144-11e5-ba71-058fbc01cf0b	2	fd40f822-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce24d572-7144-11e5-ba71-058fbc01cf0b	2	fd4107cc-7144-11e5-ba71-058fbc01cf0b	Golden Retriever

CODE

%%sql SELECT DISTINCT d.user_guid AS UserID, u.membership_type, d.dog_guid AS DogID, d.breed FROM dogs d, users u, complete_tests c WHERE d.dog_guid=c.dog_guid AND d.user_guid=u.user_guid AND d.breed="Golden Retriever";

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 711 rows affected.

CODE

%%sql SELECT DISTINCT d.user_guid AS UserID, d.dog_guid AS DogID, u.membership_type, d.breed, test_name FROM dogs d, users u, complete_tests c WHERE d.dog_guid=c.dog_guid AND d.user_guid=u.user_guid AND d.breed="golden retriever";

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 7242 rows affected.

%%sql SELECT d.user_guid AS UserID, d.dog_guid AS DogID, u.membership_type, d.breed, c.test_name FROM dogs d, users u, complete_tests c WHERE d.dog_guid=c.dog_guid AND d.user_guid=u.user_guid AND d.breed="golden retriever"

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 38268 rows affected

Practice inner joining your own tables!

Question 7: How many unique Golden Retrievers who live in North Carolina are there in the Dognition database (you should get 30)?

In [13]:

```
%%sql
SELECT u.state AS State, d.breed AS breed, COUNT(DISTINCT d.dog_guid)
FROM dogs d, users u
WHERE d.user_guid=u.user_guid AND breed="golden retriever"
GROUP BY state
HAVING state="NC";
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[13]:

State	breed	COUNT(DISTINCT d.dog_guid)
NC	Golden Retriever	30

Question 8: How many unique customers within each membership type provided reviews (there should be 2900 in the membership type with the greatest number of customers, and 15 in the membership type with the fewest number of customers if you do NOT include entries with NULL values in their ratings field)?

In [15]:

```
%%sql
SELECT u.membership_type AS MEMBERSHIP, COUNT(DISTINCT r.user_guid)
FROM users u, reviews r
WHERE u.user_guid=r.user_guid AND r.rating IS NOT NULL
GROUP BY membership_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
5 rows affected.
```

Out[15]:

MEMBERSHIP	COUNT(DISTINCT r.user_guid)
1	2900
2	1120
3	238
4	816
5	15

Question 9: For which 3 dog breeds do we have the greatest amount of site_activity data, (as defined by non-NULL values in script_detail_id) (your answers should be "Mixed", "Labrador Retriever", and "Labrador Retriever-Golden Retriever Mix"?

In [34]:

```
%%sql
SELECT d.breed, COUNT(s.script_detail_id) AS Activity
FROM dogs d, site_activities s
WHERE d.dog_guid=s.dog_guid AND s.script_detail_id IS NOT NULL
GROUP BY breed
ORDER BY activity DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[34]:

breed	Activity
Mixed	93415
Labrador Retriever	38804
Labrador Retriever-Golden Retriever Mix	27498
Golden Retriever	19802
Poodle	14938
German Shepherd Dog	13780
Golden Retriever-Labrador Retriever Mix	10980
I Don't Know	10518
Border Collie	8403
Australian Shepherd	6040

CODE

%%sql SELECT d.breed, COUNT(s.script_detail_id) AS Activity FROM dogs d, site_activities s WHERE d.dog_guid=s.dog_guid AND s.script_detail_id IS NOT NULL GROUP BY breed ORDER BY activity DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 758 rows affected.

Practice any other inner joins you would like to try here!

In [ ]:

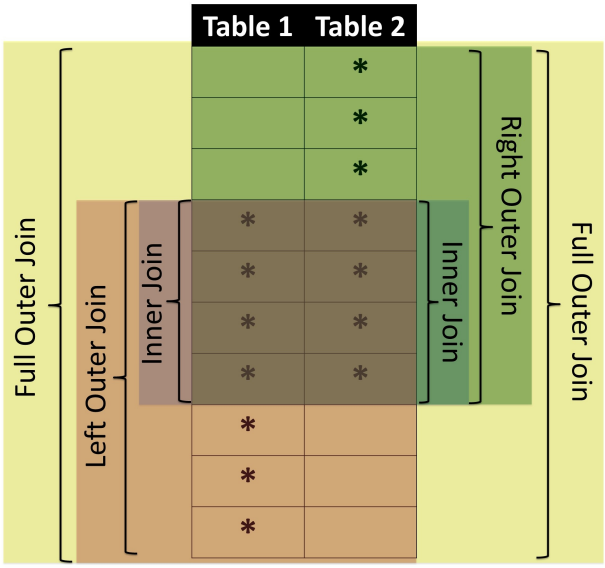
MySQL Exercise 8: Joining Tables with Outer Joins

We focused on inner joins in the last set of exercises. Most of the time inner joins will give you the results you are looking for. Occasionally, though, you might want to include all the data from a table in your calculations or your output, even if those data do not all match up with the data from the other tables you are joining with.

For example, if you have a table with customer demographic information and a table with information about which customers were sent a free sample, your might want to analyze the characteristics of customers who did and did not receive the sample. The best way to do that in a program like Tableau would be to have all your customer information in one table with an extra column indicating whether the customer received the free sample or not. Alternatively, you might want to generate a list of customers who did *not* receive the free sample, so that you can arrange for the customers to receive a free sample in the future.

In these types of situations, you will use outer joins to connect tables. Outer joins include left joins, right joins, or full outer joins (recall that full outer joins are NOT supported in MySQL). Refer to the videos "What are Joins?" and "Joins With Many to Many Relationships and Duplicates" for more information about joins.

Here's a picture to remind you of the general concepts behind outer joins:



To begin practicing outer joins, load the sql library, connect to the Dognition database, and make the Dognition database your default database:

In [4]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[4]:

```
[]
```

Left and Right Joins

Left and right joins use a different sytax than we used in the lesson about inner joins. The method I showed you to execute inner joins tells the database how to relate tables in a WHERE clause like this:

```
WHERE d.dog_guid=r.dog_guid
```

I find this syntax -- called the "equijoin" syntax -- to be very intuitive, so I thought it would be a good idea to start with it. However, we can re-write the inner joins in the same syntax used by outer joins. To use this more traditional syntax, you have to tell the database how to connect the tables using an ON clause that comes right after the FROM clause. Make sure to specify the word "JOIN" explicitly. This traditional version of the syntax frees up the WHERE clause for other things you might want to include in your query. Here's what one of our queries from the inner join lesson would look like using the traditional syntax:

```
SELECT d.dog_guid AS DogID, d.user_guid AS UserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatin
gs, d.breed, d.breed_group, d.breed_type
FROM dogs d JOIN reviews r
  ON d.dog_guid=r.dog_guid AND d.user_guid=r.user_guid
GROUP BY d.user_guid
HAVING NumRatings > 9
ORDER BY AvgRating DESC
LIMIT 200
```

You could also write "INNER JOIN" instead of "JOIN" but the default in MySQL is that JOIN will mean inner join, so including the word "INNER" is optional.

If you need a WHERE clause in the query above, it would go after the ON clause and before the GROUP BY clause.

Here's an example of a different query we used in the last lesson that employed the equijoin syntax:

```
SELECT d.user_guid AS UserID, d.dog_guid AS DogID,
       d.breed, d.breed_type, d.breed_group
FROM dogs d, complete_tests c
WHERE d.dog_guid=c.dog_guid AND test_name='Yawn Warm-up';
```

Question 1: How would you re-write this query using the traditional join syntax?

In [5]:

```
%%sql
SELECT d.dog_guid AS UserID, d.user_guid AS UserID, d.breed, d.breed_type, d.breed_group
FROM dogs d JOIN complete_tests c
  ON d.dog_guid=c.dog_guid
WHERE test_name="Yawn Warm-up"
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[5]:

UserID	UserID_1	breed	breed_type	breed_group
fd27b86c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	Pure Breed	Toy
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	ce135f2c-7144-11e5-ba71-058fbc01cf0b	Mixed	Mixed Breed/ Other/ I Don't Know	None
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	Pure Breed	Toy
fd27c5be-7144-11e5-ba71-058fbc01cf0b	ce136ac6-7144-11e5-ba71-058fbc01cf0b	Shih Tzu-Poodle Mix	Cross Breed	None
fd27b948-7144-11e5-ba71-058fbc01cf0b	ce13615c-7144-11e5-ba71-058fbc01cf0b	Siberian Husky	Pure Breed	Working
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog	Pure Breed	Herding
fd27c852-7144-11e5-ba71-058fbc01cf0b	ce136ee0-7144-11e5-ba71-058fbc01cf0b	Pug	Pure Breed	Toy
fd27c8d4-7144-11e5-ba71-058fbc01cf0b	ce136f94-7144-11e5-ba71-058fbc01cf0b	Boxer	Pure Breed	Working
fd27cf28-7144-11e5-ba71-058fbc01cf0b	ce13750c-7144-11e5-ba71-058fbc01cf0b	Chesapeake Bay Retriever	Pure Breed	Sporting
fd27c7d0-7144-11e5-ba71-058fbc01cf0b	ce136e36-7144-11e5-ba71-058fbc01cf0b	Vizsla	Pure Breed	Sporting

CODE

%%sql SELECT d.dog_guid AS UserID, d.user_guid AS UserID, d.breed, d.breed_type, d.breed_group FROM dogs d JOIN complete_tests c ON d.dog_guid=c.dog_guid WHERE test_name="Yawn Warm-up";

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 20845 rows affected.

Now that we've seen the join syntax, we can begin practicing outer joins. Our Dognition data set will make outer joins more challenging than usual, due to the lack of declared primary keys in the original database, the many-to-many relationships, and the presence of duplicate rows and NULL values in columns we will be using to combine tables. Mastering outer joins in this challenging context, though, will ensure that you understand the fundamental concepts behind joins, which will be a terrific benefit when you start writing queries in other company databases.

The first query I described above was originally written to address the question of whether dog owners who are particularly surprised by their dog's performance on Dognition tests tend to own similar breeds of dogs. When we designed this query in the last lesson, we wanted to focus on the dog owners who reported the highest average amount of surprise at their dog's performance, and provided at least 10 ratings for one or more of their dogs in the ratings. We also wanted to examine the breed, breed_type, and breed_group of each of these owner's dogs.

In examining the query, we learned that:

- All of the user_guids in the reviews table are in the dogs table
- Only 389 of the over 5000 dog_guids in the reviews table are in the dogs table

The inner join we executed resulted in 389 rows of output, because it only included the data from rows that have equivalent values in both tables being joined. But what if we wanted the full list of dogs in the reviews table and an indication of whether or not they were in the dogs table, rather than only a list of review information from dogs in the dogs table? To achieve this list, we could execute an outer join.

Let's start by using a left outer join to get the list we want. When we use the traditional join syntax to write inner joins, the order you enter the tables in your query doesn't matter. In outer joins, however, the order matters a lot. A left outer join will include all of the rows of the table to the left of the LEFT JOIN keywords. A right outer join will include all of the rows of the table to the right of the RIGHT JOIN keywords. So in order to retrieve a full list of dogs who completed at least 10 tests in the reviews table, and include as much breed information as possible, we could query:

```
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.r
ating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM reviews r LEFT JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE r.dog_guid IS NOT NULL
GROUP BY r.dog_guid
HAVING NumRatings >= 10
ORDER BY AvgRating DESC;
```

Question 2: How could you retrieve this same information using a RIGHT JOIN?

In [8]:

```
%%sql
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.rating)
AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM reviews r LEFT JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE r.dog_guid IS NOT NULL
GROUP BY r.dog_guid
HAVING NumRatings >=10
ORDER BY AvgRating DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[8]:

rDogID	dDogID	rUserID	dUserID	AvgRating	NumRatings	breed	breed_group	breed_type
fdbf39f8-7144-11e5-ba71-058fbc01cf0b	fdbf39f8-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	8.0000	12	Canaan Dog	Herding	Pure Breed
ce47553e-7144-11e5-ba71-058fbc01cf0b	None	ce6ca9ba-7144-11e5-ba71-058fbc01cf0b	None	7.8750	16	None	None	None
ce6f07e6-7144-11e5-ba71-058fbc01cf0b	None	ce7091e2-7144-11e5-ba71-058fbc01cf0b	None	7.5000	10	None	None	None
ce45ae5a-7144-11e5-ba71-058fbc01cf0b	None	ce67562c-7144-11e5-ba71-058fbc01cf0b	None	7.3529	17	None	None	None
ce2a68ac-7144-11e5-ba71-058fbc01cf0b	None	ce2a45c0-7144-11e5-ba71-058fbc01cf0b	None	7.1333	15	None	None	None
ce72be0e-7144-11e5-ba71-058fbc01cf0b	None	ce73f7a6-7144-11e5-ba71-058fbc01cf0b	None	7.0000	12	None	None	None
ce93d206-7144-11e5-ba71-058fbc01cf0b	None	ce8be19a-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None	None	None
ce97cd7a-7144-11e5-ba71-058fbc01cf0b	None	ce948926-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None	None	None
ce7038dc-7144-11e5-ba71-058fbc01cf0b	None	ce71a230-7144-11e5-ba71-058fbc01cf0b	None	6.6667	18	None	None	None
ce7dc3da-7144-11e5-ba71-058fbc01cf0b	None	ce7cbba2-7144-11e5-ba71-058fbc01cf0b	None	6.3750	16	None	None	None

CODE

%%sql SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type FROM reviews r LEFT JOIN dogs d ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid WHERE r.dog_guid IS NOT NULL GROUP BY r.dog_guid HAVING NumRatings >=10 ORDER BY AvgRating DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 932 rows affected.

QUESTION 1 ANSWER

In [14]:

```
%%sql
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.rating)
AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM dogs d RIGHT JOIN reviews r
  ON d.dog_guid=r.dog_guid AND d.user_guid=r.user_guid
WHERE r.dog_guid IS NOT NULL
GROUP BY r.dog_guid
HAVING NumRatings >=10
ORDER BY AvgRating DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[14]:

rDogID	dDogID	rUserID	dUserID	AvgRating	NumRatings	breed	breed_group	breed_type
fdbf39f8-7144-11e5-ba71-058fbc01cf0b	fdbf39f8-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	ce987914-7144-11e5-ba71-058fbc01cf0b	8.0000	12	Canaan Dog	Herding	Pure Breed
ce47553e-7144-11e5-ba71-058fbc01cf0b	None	ce6ca9ba-7144-11e5-ba71-058fbc01cf0b	None	7.8750	16	None	None	None
ce6f07e6-7144-11e5-ba71-058fbc01cf0b	None	ce7091e2-7144-11e5-ba71-058fbc01cf0b	None	7.5000	10	None	None	None
ce45ae5a-7144-11e5-ba71-058fbc01cf0b	None	ce67562c-7144-11e5-ba71-058fbc01cf0b	None	7.3529	17	None	None	None
ce2a68ac-7144-11e5-ba71-058fbc01cf0b	None	ce2a45c0-7144-11e5-ba71-058fbc01cf0b	None	7.1333	15	None	None	None
ce72be0e-7144-11e5-ba71-058fbc01cf0b	None	ce73f7a6-7144-11e5-ba71-058fbc01cf0b	None	7.0000	12	None	None	None
ce93d206-7144-11e5-ba71-058fbc01cf0b	None	ce8be19a-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None	None	None
ce97cd7a-7144-11e5-ba71-058fbc01cf0b	None	ce948926-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None	None	None
ce7038dc-7144-11e5-ba71-058fbc01cf0b	None	ce71a230-7144-11e5-ba71-058fbc01cf0b	None	6.6667	18	None	None	None
ce7dc3da-7144-11e5-ba71-058fbc01cf0b	None	ce7cbba2-7144-11e5-ba71-058fbc01cf0b	None	6.3750	16	None	None	None

CODE

%%sql SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type FROM dogs d RIGHT JOIN reviews r ON d.dog_guid=r.dog_guid AND d.user_guid=r.user_guid WHERE r.dog_guid IS NOT NULL GROUP BY r.dog_guid HAVING NumRatings >=10 ORDER BY AvgRating DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 932 rows affected.

Notice in the output of both the left and the right version of the outer join, all the rows that had a dog_guid in the reviews table but did NOT have a matching dog_guid in the dogs table have the word "None" entered in output columns related to the dogs table. "None", in this case, is Jupyter's way of saying the value is NULL. This becomes clear when you query a list of only the dog_guids that were NOT in the dogs table:

```
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.r
ating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM reviews r LEFT JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE d.dog_guid IS NULL
GROUP BY r.dog_guid
HAVING NumRatings >= 10
ORDER BY AvgRating DESC;
```

Go ahead and try it yourself (you should get 894 rows in your query):

In [18]:

```
%%sql
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.rating)
AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM reviews r LEFT JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE d.dog_guid IS NULL
GROUP BY r.dog_guid
HAVING NumRatings >=10
ORDER BY AvgRating DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[18]:

rDogID	dDogID	rUserID	dUserID	AvgRating	NumRatings	breed	breed_group	breed_type
ce47553e-7144-11e5-ba71-058fbc01cf0b	None	ce6ca9ba-7144-11e5-ba71-058fbc01cf0b	None	7.8750	16	None	None	None
ce6f07e6-7144-11e5-ba71-058fbc01cf0b	None	ce7091e2-7144-11e5-ba71-058fbc01cf0b	None	7.5000	10	None	None	None
ce45ae5a-7144-11e5-ba71-058fbc01cf0b	None	ce67562c-7144-11e5-ba71-058fbc01cf0b	None	7.3529	17	None	None	None
ce2a68ac-7144-11e5-ba71-058fbc01cf0b	None	ce2a45c0-7144-11e5-ba71-058fbc01cf0b	None	7.1333	15	None	None	None
ce72be0e-7144-11e5-ba71-058fbc01cf0b	None	ce73f7a6-7144-11e5-ba71-058fbc01cf0b	None	7.0000	12	None	None	None
ce93d206-7144-11e5-ba71-058fbc01cf0b	None	ce8be19a-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None	None	None
ce97cd7a-7144-11e5-ba71-058fbc01cf0b	None	ce948926-7144-11e5-ba71-058fbc01cf0b	None	6.7273	11	None	None	None
ce7038dc-7144-11e5-ba71-058fbc01cf0b	None	ce71a230-7144-11e5-ba71-058fbc01cf0b	None	6.6667	18	None	None	None
ce7dc3da-7144-11e5-ba71-058fbc01cf0b	None	ce7cbba2-7144-11e5-ba71-058fbc01cf0b	None	6.3750	16	None	None	None
ce866fda-7144-11e5-ba71-058fbc01cf0b	None	ce80c12a-7144-11e5-ba71-058fbc01cf0b	None	6.3000	10	None	None	None

CODE

%%sql SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.rating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type FROM reviews r LEFT JOIN dogs d ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid WHERE d.dog_guid IS NULL GROUP BY r.dog_guid HAVING NumRatings >=10 ORDER BY AvgRating DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 894 rows affected.

In order to make it easier to practice SQL queries with meaningful examples before we learned how to join tables, I added extra columns to the "dogs" table that were not in the original Dognition database. These extra columns included the "total_tests_completed" field and multiple inter-test-interval ("iti") summary fields. **Please do NOT try to use these extra fields in the query exercises below. Since you now know how to join tables, we will practice writing queries as if you only had the data provided in the original Dognition database.**

Question 3: How would you use a left join to retrieve a list of all the unique dogs in the dogs table, and retrieve a count of how many tests each one completed? Include the dog_guids and user_guids from the dogs and complete_tests tables in your output. (If you do not limit your query, your output should contain 35050 rows. HINT: use the dog_guid from the dogs table to group your results.)

In [20]:

```
%%sql
SELECT d.dog_guid AS dDogID, c.dog_guid AS cDogID, d.user_guid AS dUserID, c.user_guid AS cUserID,  COUNT(test_na
me)
FROM dogs d LEFT JOIN complete_tests c
  ON d.dog_guid=c.dog_guid
GROUP BY d.dog_guid
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[20]:

dDogID	cDogID	dUserID	cUserID	COUNT(test_name)
fd27b272-7144-11e5-ba71-058fbc01cf0b	fd27b272-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	None	21
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	None	20
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b	None	2
fd27b79a-7144-11e5-ba71-058fbc01cf0b	fd27b79a-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b	None	11
fd27b86c-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	None	31
fd27b948-7144-11e5-ba71-058fbc01cf0b	fd27b948-7144-11e5-ba71-058fbc01cf0b	ce13615c-7144-11e5-ba71-058fbc01cf0b	None	20
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	None	27
fd27baec-7144-11e5-ba71-058fbc01cf0b	None	ce1362ba-7144-11e5-ba71-058fbc01cf0b	None	0
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	fd27bbbe-7144-11e5-ba71-058fbc01cf0b	ce135f2c-7144-11e5-ba71-058fbc01cf0b	None	20
fd27be84-7144-11e5-ba71-058fbc01cf0b	None	ce13697c-7144-11e5-ba71-058fbc01cf0b	None	0

CODE

%%sql SELECT d.dog_guid AS dDogID, c.dog_guid AS cDogID, d.user_guid AS dUserID, c.user_guid AS cUserID, COUNT(test_name) FROM dogs d LEFT JOIN complete_tests c ON d.dog_guid=c.dog_guid GROUP BY d.dog_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 35050 rows affected.

Sometimes you can get so focused on writing your join statement that you don't pay close attention to the fields and tables you put in your other clauses, especially when you are joining a lot of tables. Often your query will still run successfully, even if you haven't entered the criteria or grouping clause you intended. The next question will illustrate how easy it is for this to happen.

Question 4: Repeat the query you ran in Question 3, but intentionally use the dog_guids from the completed_tests table to group your results instead of the dog_guids from the dogs table. (Your output should contain 17987 rows)

In [23]:

```
%%sql
SELECT d.dog_guid AS dDogID, c.dog_guid AS cDogID, d.user_guid AS dUserID, c.user_guid AS cUserID,  COUNT(test_na
me)
FROM dogs d LEFT JOIN complete_tests c
  ON d.dog_guid=c.dog_guid
GROUP BY c.dog_guid
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[23]:

dDogID	cDogID	dUserID	cUserID	COUNT(test_name)
fd27baec-7144-11e5-ba71-058fbc01cf0b	None	ce1362ba-7144-11e5-ba71-058fbc01cf0b	None	0
fd27b272-7144-11e5-ba71-058fbc01cf0b	fd27b272-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	None	21
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	None	20
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b	None	2
fd27b79a-7144-11e5-ba71-058fbc01cf0b	fd27b79a-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b	None	11
fd27b86c-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	None	31
fd27b948-7144-11e5-ba71-058fbc01cf0b	fd27b948-7144-11e5-ba71-058fbc01cf0b	ce13615c-7144-11e5-ba71-058fbc01cf0b	None	20
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	None	27
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	fd27bbbe-7144-11e5-ba71-058fbc01cf0b	ce135f2c-7144-11e5-ba71-058fbc01cf0b	None	20
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	fd27c1c2-7144-11e5-ba71-058fbc01cf0b	ce136a1c-7144-11e5-ba71-058fbc01cf0b	None	20

CODE

%%sql SELECT d.dog_guid AS dDogID, c.dog_guid AS cDogID, d.user_guid AS dUserID, c.user_guid AS cUserID, COUNT(test_name) FROM dogs d LEFT JOIN complete_tests c ON d.dog_guid=c.dog_guid GROUP BY c.dog_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 17987 rows affected.

This time your query ran successfully, but you retrieved many fewer DogIDs because the GROUP BY clause grouped your results according to the dog_guids in the completed_tests table rather than the dog_guid table. As a result, even though you implemented your join correctly, all of the dog_guids that were in the dogs table but not in the completed_tests table got rolled up into one row of your output where completed_tests.dogs_guid = NULL. This is a good opportunity to remind ourselves about the differences between SELECT/GROUP BY and COUNT DISTINCT.

Question 5: Write a query using COUNT DISTINCT to determine how many distinct dog_guids there are in the completed_tests table.

In [24]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM complete_tests
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[24]:

COUNT(DISTINCT dog_guid)
17986

CODE

%%sql SELECT d.dog_guid AS dDogID, c.dog_guid AS cDogID, d.user_guid AS dUserID, c.user_guid AS cUserID, COUNT(test_name) FROM dogs d LEFT JOIN complete_tests c ON d.dog_guid=c.dog_guid WHERE c.dog_guid IS NOT NULL GROUP BY c.dog_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 17986 rows affected.

In [8]:

```
%%sql
SELECT d.dog_guid AS dDogID, c.dog_guid AS cDogID, d.user_guid AS dUserID, c.user_guid AS cUserID,  COUNT(test_na
me)
FROM dogs d LEFT JOIN complete_tests c
  ON d.dog_guid=c.dog_guid
WHERE d.dog_guid IS NOT NULL
GROUP BY d.dog_guid
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[8]:

dDogID	cDogID	dUserID	cUserID	COUNT(test_name)
fd27b272-7144-11e5-ba71-058fbc01cf0b	fd27b272-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	None	21
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	None	20
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b	None	2
fd27b79a-7144-11e5-ba71-058fbc01cf0b	fd27b79a-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b	None	11
fd27b86c-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	None	31
fd27b948-7144-11e5-ba71-058fbc01cf0b	fd27b948-7144-11e5-ba71-058fbc01cf0b	ce13615c-7144-11e5-ba71-058fbc01cf0b	None	20
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	None	27
fd27baec-7144-11e5-ba71-058fbc01cf0b	None	ce1362ba-7144-11e5-ba71-058fbc01cf0b	None	0
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	fd27bbbe-7144-11e5-ba71-058fbc01cf0b	ce135f2c-7144-11e5-ba71-058fbc01cf0b	None	20
fd27be84-7144-11e5-ba71-058fbc01cf0b	None	ce13697c-7144-11e5-ba71-058fbc01cf0b	None	0
fd27bf60-7144-11e5-ba71-058fbc01cf0b	None	ce1352ac-7144-11e5-ba71-058fbc01cf0b	None	0
fd27c032-7144-11e5-ba71-058fbc01cf0b	None	ce1352ac-7144-11e5-ba71-058fbc01cf0b	None	0
fd27c0fa-7144-11e5-ba71-058fbc01cf0b	None	ce136a1c-7144-11e5-ba71-058fbc01cf0b	None	0
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	fd27c1c2-7144-11e5-ba71-058fbc01cf0b	ce136a1c-7144-11e5-ba71-058fbc01cf0b	None	20
fd27c294-7144-11e5-ba71-058fbc01cf0b	None	ce1366e8-7144-11e5-ba71-058fbc01cf0b	None	0
fd27c35c-7144-11e5-ba71-058fbc01cf0b	None	ce1366e8-7144-11e5-ba71-058fbc01cf0b	None	0
fd27c424-7144-11e5-ba71-058fbc01cf0b	None	ce1366e8-7144-11e5-ba71-058fbc01cf0b	None	0
fd27c4ec-7144-11e5-ba71-058fbc01cf0b	None	ce1366e8-7144-11e5-ba71-058fbc01cf0b	None	0
fd27c5be-7144-11e5-ba71-058fbc01cf0b	fd27c5be-7144-11e5-ba71-058fbc01cf0b	ce136ac6-7144-11e5-ba71-058fbc01cf0b	None	20
fd27c64a-7144-11e5-ba71-058fbc01cf0b	None	ce136c24-7144-11e5-ba71-058fbc01cf0b	None	0

CODE

%%sql SELECT d.dog_guid AS dDogID, c.dog_guid AS cDogID, d.user_guid AS dUserID, c.user_guid AS cUserID, COUNT(test_name) FROM dogs d LEFT JOIN complete_tests c ON d.dog_guid=c.dog_guid WHERE d.dog_guid IS NOT NULL GROUP BY d.dog_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 35050 rows affected.

The result of your COUNT DISTINCT clause should be 17,986 which is one row less than the number of rows you retrieved from your query in Question 4. That's because COUNT DISTINCT does NOT count NULL values, while SELECT/GROUP BY clauses roll up NULL values into one group. If you want to infer the number of distinct entries from the results of a query using joins and GROUP BY clauses, remember to include an "IS NOT NULL" clause to ensure you are not counting NULL values.

These exercises are a good illustration of why it is very helpful to save your queries when you are doing an analysis. Saving your queries allows you and your team members to double-check your work later. As you can see, the concepts behind SQL aren't themselves too tricky, but it is easy to make mistakes, especially when your queries get long and more complicated.

One more situation where joins can cause some confusion is when you have duplicate rows in a table you are joining. If you ignore what we've discussed about set theory and the way databases compute their joins, the behavior databases exhibit when you have duplicate rows in a joined table will seem utterly baffling. With this knowledge, though, the behavior will make perfect sense. Let's walk through what happens.

Question 6: We want to extract all of the breed information of every dog a user_guid in the users table owns. If a user_guid in the users table does not own a dog, we want that information as well. Write a query that would return this information. Include the dog_guid from the dogs table, and user_guid from both the users and dogs tables in your output. (HINT: you should get 952557 rows in your output!)

In [10]:

```
%%sql
SELECT u.user_guid AS uUserID, d.user_guid AS dUserID, d.dog_guid AS dDogID,
d.breed
FROM users u LEFT JOIN dogs d
ON u.user_guid=d.user_guid
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[10]:

uUserID	dUserID	dDogID	breed
ce134e42-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	fd27b272-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever
ce134e42-7144-11e5-ba71-058fbc01cf0b	ce134e42-7144-11e5-ba71-058fbc01cf0b	fd417cac-7144-11e5-ba71-058fbc01cf0b	Mixed
ce1353d8-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	fd27b5ba-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog
ce1353d8-7144-11e5-ba71-058fbc01cf0b	ce1353d8-7144-11e5-ba71-058fbc01cf0b	fd3fb0f2-7144-11e5-ba71-058fbc01cf0b	Shetland Sheepdog
ce135ab8-7144-11e5-ba71-058fbc01cf0b	ce135ab8-7144-11e5-ba71-058fbc01cf0b	fd27b6b4-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce13507c-7144-11e5-ba71-058fbc01cf0b	ce13507c-7144-11e5-ba71-058fbc01cf0b	fd27b79a-7144-11e5-ba71-058fbc01cf0b	Golden Retriever
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ba1a-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27e9a4-7144-11e5-ba71-058fbc01cf0b	Shih Tzu
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27ed46-7144-11e5-ba71-058fbc01cf0b	Shih Tzu

CODE

%%sql SELECT u.user_guid AS uUserID, d.user_guid AS dUserID, d.dog_guid AS dDogID, d.breed FROM dogs d RIGHT JOIN users u ON d.user_guid=u.user_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 952557 rows affected.

There are only 35050 distinct dog_guids in the dogs table. Why is the database outputting almost a million rows? That can't be right. Let's figure out what is going on.

Question 7: Adapt the query you wrote above so that it counts the number of rows the join will output per user_id. Sort the results by this count in descending order. Remember that if you include dog_guid or breed fields in this query, they will be randomly populated by only one of the values associated with a user_guid (see MySQL Exercise 6; there should be 33,193 rows in your output).

In [5]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM dogs
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[5]:

COUNT(DISTINCT dog_guid)
35050

In [7]:

```
%%sql
SELECT u.user_guid AS uUserID, d.user_guid AS dUserID, d.dog_guid AS dDogID, d.breed, count(*) AS numrows
FROM users u LEFT JOIN dogs d
  ON u.user_guid=d.user_guid
GROUP BY u.user_guid
ORDER BY numrows DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[7]:

uUserID	dUserID	dDogID	breed	numrows
ce7b75bc-7144-11e5-ba71-058fbc01cf0b	ce7b75bc-7144-11e5-ba71-058fbc01cf0b	fd7bfb52-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	913138
ce225842-7144-11e5-ba71-058fbc01cf0b	ce225842-7144-11e5-ba71-058fbc01cf0b	fd423714-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	442
ce2258a6-7144-11e5-ba71-058fbc01cf0b	ce2258a6-7144-11e5-ba71-058fbc01cf0b	fd40bd62-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	320
ce135e14-7144-11e5-ba71-058fbc01cf0b	ce135e14-7144-11e5-ba71-058fbc01cf0b	fd27b86c-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	130
ce29675e-7144-11e5-ba71-058fbc01cf0b	ce29675e-7144-11e5-ba71-058fbc01cf0b	fd46b014-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever- Mix	110
ce6676d0-7144-11e5-ba71-058fbc01cf0b	ce6676d0-7144-11e5-ba71-058fbc01cf0b	fd68ed6e-7144-11e5-ba71-058fbc01cf0b	Golden Retriever	64
ce7adeea-7144-11e5-ba71-058fbc01cf0b	ce7adeea-7144-11e5-ba71-058fbc01cf0b	fd7b3faa-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever	49
ce47264a-7144-11e5-ba71-058fbc01cf0b	ce47264a-7144-11e5-ba71-058fbc01cf0b	fd401614-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	36
ce8c2d08-7144-11e5-ba71-058fbc01cf0b	ce8c2d08-7144-11e5-ba71-058fbc01cf0b	fdb54786-7144-11e5-ba71-058fbc01cf0b	Basenji	36
ce66713a-7144-11e5-ba71-058fbc01cf0b	ce66713a-7144-11e5-ba71-058fbc01cf0b	fd68e80a-7144-11e5-ba71-058fbc01cf0b	Puggle	30
ce964888-7144-11e5-ba71-058fbc01cf0b	ce964888-7144-11e5-ba71-058fbc01cf0b	fdbbdd4e-7144-11e5-ba71-058fbc01cf0b	Mixed	30
ce26b266-7144-11e5-ba71-058fbc01cf0b	ce26b266-7144-11e5-ba71-058fbc01cf0b	fd436e7c-7144-11e5-ba71-058fbc01cf0b	Pug	25
ce6e67e6-7144-11e5-ba71-058fbc01cf0b	ce6e67e6-7144-11e5-ba71-058fbc01cf0b	fd6cbb42-7144-11e5-ba71-058fbc01cf0b	Australian Shepherd	25
ce728bf0-7144-11e5-ba71-058fbc01cf0b	ce728bf0-7144-11e5-ba71-058fbc01cf0b	fd71a2f6-7144-11e5-ba71-058fbc01cf0b	Labrador Retriever-Golden Retriever Mix	25
ce135766-7144-11e5-ba71-058fbc01cf0b	ce135766-7144-11e5-ba71-058fbc01cf0b	fd3ccf2c-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	24
ce66b9b0-7144-11e5-ba71-058fbc01cf0b	ce66b9b0-7144-11e5-ba71-058fbc01cf0b	fd692aa4-7144-11e5-ba71-058fbc01cf0b	Mixed	24
ce83d2ca-7144-11e5-ba71-058fbc01cf0b	ce83d2ca-7144-11e5-ba71-058fbc01cf0b	fdad96a8-7144-11e5-ba71-058fbc01cf0b	Cockapoo	24
ce7da332-7144-11e5-ba71-058fbc01cf0b	ce7da332-7144-11e5-ba71-058fbc01cf0b	fd80bebc-7144-11e5-ba71-058fbc01cf0b	Mixed	20
ce134492-7144-11e5-ba71-058fbc01cf0b	ce134492-7144-11e5-ba71-058fbc01cf0b	fd40e206-7144-11e5-ba71-058fbc01cf0b	Shih Tzu	18
ce9a381c-7144-11e5-ba71-058fbc01cf0b	ce9a381c-7144-11e5-ba71-058fbc01cf0b	fdc12bbe-7144-11e5-ba71-058fbc01cf0b	Chihuahua	18

CODE

%%sql SELECT u.user_guid AS uUserID, d.user_guid AS dUserID, d.dog_guid AS dDogID, d.breed, count(*) AS numrows FROM users u LEFT JOIN dogs d ON u.user_guid=d.user_guid GROUP BY u.user_guid ORDER BY numrows DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 33193 rows affected.

This query told us that user 'ce7b75bc-7144-11e5-ba71-058fbc01cf0b' would be associated with 913,138 rows in the output of the outer join we designed! Once again, why? We are going to work with the second user_guid in the output you just generated, 'ce225842-7144-11e5-ba71-058fbc01cf0b', because it would be associated with 442 output rows, and 442 rows are much easier to work with than 913,138.

Question 8: How many rows in the *users* table are associated with user_guid 'ce225842-7144-11e5-ba71-058fbc01cf0b'?

In [8]:

```
%%sql
SELECT COUNT(user_guid)
FROM users
WHERE user_guid='ce225842-7144-11e5-ba71-058fbc01cf0b'
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[8]:

COUNT(user_guid)
17

There are 17 entries associated with that user_guid in the users table. If you examine all the columns in the entries, you will see that the rows are exact duplicates of each other. That's unfortunate, but also something that can happen in real life data sets, especially those from new companies or governmental agencies.

Ok, now...

Question 9: Examine all the rows in the *dogs* table that are associated with user_guid 'ce225842-7144-11e5-ba71-058fbc01cf0b'?

In [9]:

```
%%sql
SELECT *
FROM dogs
WHERE user_guid='ce225842-7144-11e5-ba71-058fbc01cf0b'
```

```
 * mysql://studentuser:***@localhost/dognitiondb
26 rows affected.
```

Out[9]:

gender	birthday	breed	weight	dog_fixed	dna_tested	created_at	updated_at	dimension	exclude	breed_type	breed_group	d
male	2013	Shih Tzu	190	1	1	2013-03-21 19:30:06	2015-05-13 18:01:39	maverick	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	1	1	2013-03-30 19:41:08	2015-01-06 16:26:41	stargazer	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	1	1	2013-04-08 15:44:08	2015-09-21 18:05:51	einstein	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	1	1	2013-05-03 19:56:17	2013-07-25 19:42:39	None	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	1	1	2013-05-16 16:03:54	2014-10-20 18:10:48	None	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	1	1	2013-05-17 16:03:57	2014-11-05 16:18:29	stargazer	1	Pure Breed	Toy	71 058f
male	2013	Shih Tzu	190	1	1	2013-05-23 12:29:28	2014-10-06 19:29:11	None	1	Pure Breed	Toy	f 71 058f
female	2013	Shih Tzu	190	1	1	2013-07-18 19:16:30	2014-05-28 14:11:10	None	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	0	1	1	2013-07-30 19:39:37	2013-07-30 19:39:37	None	None	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	1	1	2013-08-05 19:50:55	2014-07-20 15:06:18	None	1	Pure Breed	Toy	f 71 058f
female	2013	Shih Tzu	190	1	1	2013-07-18 17:44:47	2013-10-31 18:32:39	None	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	0	0	2013-09-24	2014-05-28	None	1	Pure Breed	Toy	71

						18:09:10	14:09:36					058f
female	2012	Spinone Italiano	50	0	0	2013-09-29 14:56:10	2014-06-13 15:13:14	None	None	Pure Breed	Sporting	f 71 058f
male	2013	Shih Tzu	190	0	0	2013-10-08 21:24:19	2014-05-28 14:09:05	None	1	Pure Breed	Toy	f 71 058f
female	2012	Shih Tzu	190	0	0	2013-12-03 18:24:30	2014-05-28 14:08:56	None	1	Pure Breed	Toy	f 71 058f
male	2009	Shih Tzu	190	0	0	2013-12-19 19:21:14	2014-05-28 14:09:49	None	1	Pure Breed	Toy	f 71 058f
male	2010	Shih Tzu	190	0	0	2014-04-29 14:12:54	2014-06-13 15:13:15	None	1	Pure Breed	Toy	f 71 058f
male	2014	Shih Tzu	190	0	0	2014-05-22 21:05:33	2014-07-10 13:57:17	None	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	0	None	2014-06-17 20:15:38	2014-06-30 22:28:42	stargazer	1	Pure Breed	Toy	f 71 058f
male	2014	Shih Tzu	190	None	None	2014-07-02 14:04:56	2014-07-02 14:04:56	None	None	Pure Breed	Toy	f 71 058f
male	2014	Shih Tzu	190	0	0	2014-05-07 18:38:59	2014-05-28 14:06:17	None	1	Pure Breed	Toy	f 71 058f
male	2009	Shih Tzu	190	0	0	2014-05-07 20:11:13	2014-05-28 14:04:53	None	1	Pure Breed	Toy	f 71 058f
male	2013	Shih Tzu	190	0	0	2014-05-07 20:32:33	2014-05-28 14:03:55	None	1	Pure Breed	Toy	f 71 058f
female	2011	Akita-Airedale Terrier Mix	20	0	None	2014-10-06 02:46:31	2014-10-06 02:46:31	None	None	Cross Breed	None	f 71 058f
male	2014	Australian Terrier	150	0	None	2015-04-13 14:40:48	2015-04-13 14:40:48	None	None	Pure Breed	Terrier	71 058f
male	2014	Shih Tzu	190	0	None	2015-04-27 15:29:36	2015-04-27 15:29:36	None	None	Pure Breed	Toy	f 71

You should see there are 26 rows associated with that UserID in the dogs table. When you examine the dogs table, you see that there are a lot of entries that have "Shih Tzu" in the breed column and "190" in the weight column. This was Dognition's internal convention for indicating test accounts. So these dog_guids and user_guids do not represent real data. Nonetheless, they provide a great example of what happens when you join on fields that have duplicate entries.

Recall the general strategy relational databases use to join tables:

Set 1

EmployeeID	Name
AA	Lucy
BB	Joe
CC	Luke

✖

Set 2

DepartID	Department
1	Clothing
2	Electronics

Cartesian (Cross) Product

AA	Lucy	1	Clothing
AA	Lucy	2	Electronics
BB	Joe	1	Clothing
BB	Joe	2	Electronics
CC	Luke	1	Clothing
CC	Luke	2	Electronics

When databases join tables, they output the result of every pair of entries that meet certain criteria in the linking column of one table with the linking column of another table. Our join statement imposed the criteria that the output should only include pairs whose user_guids matched in the two linking columns. However, since there were multiple rows that had the same user_guid in the users table, *each one of these rows got paired up with each row in the dogs table that had the same user_guid.* The result was 442 rows, because 17 (instances of the user_guid in the users table) x 26 (instances of the user_guid in the dogs table) = 442.

Having seen this, perhaps you can now appreciate why some database experts emphasize terminology that differentiates between set *theory* and real database *implementation*. Database operations, like those that join tables, are based on set theory that assumes there are no duplicate rows in your tables. In real life, duplicate rows get entered all the time, and it can cause you to misinterpret your queries if you do not understand the consequences. If you've been impacted enough times by the differences between real and theoretical databases, it makes sense that it would be important to you to use language that clearly distinguishes between theory and real life.

The important things I want you to remember from this example of joins with duplicates are that duplicate rows and table relationships that have table-to-table mappings of greater than 1 have multiplicative effects on your query results, due to the way relational databases combine tables. If you write queries that aggregate over a lot of joined tables, it can be very difficult to catch issues that output results you don't intend, because the aggregated results will hide clues from you. To prevent this from happening, I recommend you adopt the following practices:

- Avoid making assumptions about your data or your analyses. For example, rather than assume that all the values in a column are unique just because some documentation says they should be, check for yourself!
- Always look at example outputs of your queries before you strongly interpret aggregate calculations. Take extra care to do this when your queries require joins.
- When your queries require multiple layers of functions or joins, examine the output of each layer or join first before you combine them all together.
- Adopt a healthy skepticsm of all your data and results. If you see something you don't expect, make sure you explore it before interpreting it strongly or incorporating it into other analyses.

One more type of join to mention that I discussed in the joins videos is a full outer join. Full outer joins include all of the rows in both tables in an ON clause, regardless of whether there is a value that links the row of one table with a row in the other table. As with left or right joins, whenever a value in a row does not have a matching value in the joined table, NULLs will be entered for all values in the joined table.

Outer joins are used very rarely. The most practical application is if you want to export all of your raw data to another program for visualization or analysis. The syntax for outer joins is the same as for inner joins, but you replace the word "inner" with " full outer":

```
SELECT r.dog_guid AS rDogID, d.dog_guid AS dDogID, r.user_guid AS rUserID, d.user_guid AS dUserID, AVG(r.r
ating) AS AvgRating, COUNT(r.rating) AS NumRatings, d.breed, d.breed_group, d.breed_type
FROM reviews r FULL OUTER JOIN dogs d
  ON r.dog_guid=d.dog_guid AND r.user_guid=d.user_guid
WHERE r.dog_guid IS NOT NULL
GROUP BY r.dog_guid
ORDER BY AvgRating DESC;
```

HOWEVER! MySQL does not support full outer joins.

If you wanted to imitate a full outer join in mySQL, you could follow one of the methods described at this website:

http://www.xaprb.com/blog/2006/05/26/how-to-write-full-outer-join-in-mysql/ (http://www.xaprb.com/blog/2006/05/26/how-to-write-full-outer-join-in-mysql/)

Practice outer joining your own tables!¶

Question 10: How would you write a query that used a *left* join to return the number of distinct user_guids that were in the users table, but not the dogs table (your query should return a value of 2226)?

In [5]:

```
%%sql
SELECT COUNT(DISTINCT u.user_guid)
FROM users u LEFT JOIN dogs d
ON u.user_guid=d.user_guid
WHERE d.user_guid IS NULL;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[5]:

COUNT(DISTINCT u.user_guid)
2226

Question 11: How would you write a query that used a *right* join to return the number of distinct user_guids that were in the users table, but not the dogs table (your query should return a value of 2226)?

In [6]:

```
%%sql
SELECT COUNT(DISTINCT u.user_guid)
FROM dogs d RIGHT JOIN users u
ON d.user_guid=u.user_guid
WHERE d.user_guid IS NULL;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[6]:

COUNT(DISTINCT u.user_guid)
2226

Question 12: Use a left join to create a list of all the unique dog_guids that are contained in the site_activities table, but not the dogs table, and how many times each one is entered. Note that there are a lot of NULL values in the dog_guid of the site_activities table, so you will want to exclude them from your list. (Hint: if you exclude null values, the results you get will have two rows with words in their site_activities dog_guid fields instead of real guids, due to mistaken entries)

In [7]:

```
%%sql
SELECT s.dog_guid AS SA_dogs_not_present_in_dogs_table, COUNT(*) AS
NumEntries
FROM site_activities s LEFT JOIN dogs d
ON s.dog_guid=d.dog_guid
WHERE d.dog_guid IS NULL AND s.dog_guid IS NOT NULL
GROUP BY SA_dogs_not_present_in_dogs_table;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[7]:

SA_dogs_not_present_in_dogs_table	NumEntries
Membership	5587
PortalContent	12

Practice any other outer joins you are interested in here!

In [ ]:

```

```

MySQL Exercise 10: Useful Logical Operators

There are a few more logical operators we haven't covered yet that you might find useful when designing your queries. Expressions that use logical operators return a result of "true" or "false", depending on whether the conditions you specify are met. The "true" or "false" results are usually used to determine which, if any, subsequent parts of your query will be run. We will discuss the IF operator, the CASE operator, and the order of operations within logical expressions in this lesson.

Begin by loading the sql library and database, and making the Dognition database your default database:

In [5]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[5]:

```
[]
```

1. IF expressions

IF expressions are used to return one of two results based on whether inputs to the expressions meet the conditions you specify. They are frequently used in SELECT statements as a compact way to rename values in a column. The basic syntax is as follows:

```
IF([your conditions],[value outputted if conditions are met],[value outputted if conditions are NOT met])
```

So we could write:

```
SELECT created_at, IF(created_at<'2014-06-01','early_user','late_user') AS user_type
FROM users
```

to output one column that provided the time stamp of when a user account was created, and a second column called user_type that used that time stamp to determine whether the user was an early or late user. User_type could then be used in a GROUP BY statement to segment summary calculations (in database systems that support the use of aliases in GROUP BY statements).

For example, since we know there are duplicate user_guids in the user table, we could combine a subquery with an IF statement to retrieve a list of unique user_guids with their classification as either an early or late user (based on when their first user entry was created):

```
SELECT cleaned_users.user_guid as UserID,
       IF(cleaned_users.first_account<'2014-06-01','early_user','late_user') AS user_type
FROM (SELECT user_guid, MIN(created_at) AS first_account
      FROM users
      GROUP BY user_guid) AS cleaned_users
```

We could then use a GROUP BY statement to count the number of unique early or late users:

```
SELECT IF(cleaned_users.first_account<'2014-06-01','early_user','late_user') AS user_type,
       COUNT(cleaned_users.first_account)
FROM (SELECT user_guid, MIN(created_at) AS first_account
      FROM users
      GROUP BY user_guid) AS cleaned_users
GROUP BY user_type
```

Try it yourself:

In [7]:

```
%%sql
SELECT cleaned_users.user_guid as UserID,
       IF(cleaned_users.first_account<'2014-06-01','early_user','late_user') AS user_type
FROM (SELECT user_guid, MIN(created_at) AS first_account
      FROM users
      GROUP BY user_guid) AS cleaned_users
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[7]:

UserID	user_type
ce134492-7144-11e5-ba71-058fbc01cf0b	early_user
ce134a78-7144-11e5-ba71-058fbc01cf0b	early_user
ce134be0-7144-11e5-ba71-058fbc01cf0b	early_user
ce134d16-7144-11e5-ba71-058fbc01cf0b	early_user
ce134e42-7144-11e5-ba71-058fbc01cf0b	early_user
ce134f50-7144-11e5-ba71-058fbc01cf0b	early_user
ce13507c-7144-11e5-ba71-058fbc01cf0b	early_user
ce135194-7144-11e5-ba71-058fbc01cf0b	early_user
ce1352ac-7144-11e5-ba71-058fbc01cf0b	early_user
ce1353d8-7144-11e5-ba71-058fbc01cf0b	early_user

CODE

%%sql SELECT cleaned_users.user_guid as UserID, IF(cleaned_users.first_account<'2014-06-01','early_user','late_user') AS user_type FROM (SELECT user_guid, MIN(created_at) AS first_account FROM users GROUP BY user_guid) AS cleaned_users;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 33193 rows affected.

In [8]:

```
%%sql
SELECT IF(cleaned_users.first_account<'2014-06-01','early_user','late_user') AS user_type,
       COUNT(cleaned_users.first_account)
FROM (SELECT user_guid, MIN(created_at) AS first_account
      FROM users
      GROUP BY user_guid) AS cleaned_users
GROUP BY user_type;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[8]:

user_type	COUNT(cleaned_users.first_account)
early_user	14470
late_user	18723

Question 1: Write a query that will output distinct user_guids and their associated country of residence from the users table, excluding any user_guids or countries that have NULL values. You should get 16,261 rows in your result.

In [11]:

```
%%sql
SELECT DISTINCT user_guid, country
FROM users
WHERE country IS NOT NULL
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[11]:

user_guid	country
ce134e42-7144-11e5-ba71-058fbc01cf0b	US
ce1353d8-7144-11e5-ba71-058fbc01cf0b	US
ce135ab8-7144-11e5-ba71-058fbc01cf0b	US
ce13507c-7144-11e5-ba71-058fbc01cf0b	US
ce135e14-7144-11e5-ba71-058fbc01cf0b	US
ce13615c-7144-11e5-ba71-058fbc01cf0b	US
ce135f2c-7144-11e5-ba71-058fbc01cf0b	US
ce136a1c-7144-11e5-ba71-058fbc01cf0b	US
ce136ac6-7144-11e5-ba71-058fbc01cf0b	US
ce136c24-7144-11e5-ba71-058fbc01cf0b	US

CODE

%%sql SELECT DISTINCT user_guid, country FROM users WHERE country IS NOT NULL;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 16261 rows affected.

In [14]:

```
%%sql
SELECT DISTINCT user_guid, country
FROM users
WHERE country IS NOT NULL
GROUP BY country
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[14]:

user_guid	country
ce98edd6-7144-11e5-ba71-058fbc01cf0b	AD
ce7c575c-7144-11e5-ba71-058fbc01cf0b	AE
ce32d1d6-7144-11e5-ba71-058fbc01cf0b	AR
ce257fc2-7144-11e5-ba71-058fbc01cf0b	AT
ce221d1e-7144-11e5-ba71-058fbc01cf0b	AU
ce32991e-7144-11e5-ba71-058fbc01cf0b	BA
ce3e2d10-7144-11e5-ba71-058fbc01cf0b	BE
ce728632-7144-11e5-ba71-058fbc01cf0b	BG
ce46ec16-7144-11e5-ba71-058fbc01cf0b	BM
ce252c98-7144-11e5-ba71-058fbc01cf0b	BR

CODE

%%sql SELECT DISTINCT user_guid, country FROM users WHERE country IS NOT NULL GROUP BY country;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 69 rows affected.

In [26]:

```
%%sql
SELECT DISTINCT user_guid, country, COUNT(country)
FROM users
WHERE country IS NOT NULL
GROUP BY country DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[26]:

user_guid	country	COUNT(country)
ce2b9a42-7144-11e5-ba71-058fbc01cf0b	ZA	18
ce3c3dfc-7144-11e5-ba71-058fbc01cf0b	VE	2
ce134e42-7144-11e5-ba71-058fbc01cf0b	US	10310
ce8a9e48-7144-11e5-ba71-058fbc01cf0b	UA	1
ce9122cc-7144-11e5-ba71-058fbc01cf0b	TT	1
ce7f4d22-7144-11e5-ba71-058fbc01cf0b	TR	1
ce893436-7144-11e5-ba71-058fbc01cf0b	SI	2
ce138f92-7144-11e5-ba71-058fbc01cf0b	SG	19
ce362610-7144-11e5-ba71-058fbc01cf0b	SE	9
ce286584-7144-11e5-ba71-058fbc01cf0b	SA	1
ce8b6166-7144-11e5-ba71-058fbc01cf0b	RU	4
ce280aa8-7144-11e5-ba71-058fbc01cf0b	RO	3
ce73fb98-7144-11e5-ba71-058fbc01cf0b	PT	4
ce756c3a-7144-11e5-ba71-058fbc01cf0b	PR	2
ce6756ae-7144-11e5-ba71-058fbc01cf0b	PL	5
ce265fe6-7144-11e5-ba71-058fbc01cf0b	PH	3
ce83bfa6-7144-11e5-ba71-058fbc01cf0b	PE	2
ce2241ae-7144-11e5-ba71-058fbc01cf0b	NZ	45
ce253f80-7144-11e5-ba71-058fbc01cf0b	NO	33
ce298a18-7144-11e5-ba71-058fbc01cf0b	NL	19

CODE

%%sql SELECT DISTINCT user_guid, country, COUNT(country) FROM users WHERE country IS NOT NULL GROUP BY country DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 69 rows affected.

In [27]:

```
%%sql
SELECT DISTINCT user_guid, country, COUNT(country) AS NumOfResidents
FROM users
WHERE country IS NOT NULL
GROUP BY country
ORDER BY NumOfResidents DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[27]:

user_guid	country	NumOfResidents
ce134e42-7144-11e5-ba71-058fbc01cf0b	US	10310
ce28a468-7144-11e5-ba71-058fbc01cf0b	N/A	6267
ce2209d2-7144-11e5-ba71-058fbc01cf0b	CA	551
ce221d1e-7144-11e5-ba71-058fbc01cf0b	AU	169
ce220a72-7144-11e5-ba71-058fbc01cf0b	GB	158
ce24d6e4-7144-11e5-ba71-058fbc01cf0b	DE	45
ce2241ae-7144-11e5-ba71-058fbc01cf0b	NZ	45
ce13851a-7144-11e5-ba71-058fbc01cf0b	DK	36
ce253f80-7144-11e5-ba71-058fbc01cf0b	NO	33
ce137a7a-7144-11e5-ba71-058fbc01cf0b	FR	30
ce252c98-7144-11e5-ba71-058fbc01cf0b	BR	27
ce22234a-7144-11e5-ba71-058fbc01cf0b	CH	24
ce34c3ba-7144-11e5-ba71-058fbc01cf0b	ES	23
ce26e614-7144-11e5-ba71-058fbc01cf0b	IT	23
ce298a18-7144-11e5-ba71-058fbc01cf0b	NL	19
ce138f92-7144-11e5-ba71-058fbc01cf0b	SG	19
ce2b9a42-7144-11e5-ba71-058fbc01cf0b	ZA	18
ce24bdd0-7144-11e5-ba71-058fbc01cf0b	MX	16
ce137bce-7144-11e5-ba71-058fbc01cf0b	EE	15
ce3e2d10-7144-11e5-ba71-058fbc01cf0b	BE	12

CODE

%%sql SELECT DISTINCT user_guid, country, COUNT(country) AS NumOfResidents FROM users WHERE country IS NOT NULL GROUP BY country ORDER BY NumOfResidents DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 69 rows affected.

Question 2: Use an IF expression and the query you wrote in Question 1 as a subquery to determine the number of unique user_guids who reside in the United States (abbreviated "US") and outside of the US.

In [15]:

```
%%sql
SELECT IF(cleaned_users.country='US','In US','Outside US') AS user_location,
count(cleaned_users.user_guid) AS num_guids
FROM (SELECT DISTINCT user_guid, country
FROM users
WHERE user_guid IS NOT NULL AND country IS NOT NULL) AS cleaned_users
GROUP BY user_location;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[15]:

user_location	num_guids
In US	9356
Outside US	6905

Single IF expressions can only result in one of two specified outputs, but multiple IF expressions can be nested to result in more than two possible outputs. When you nest IF expressions, it is important to encase each IF expression--as well as the entire IF expression put together--in parentheses.

For example, if you examine the entries contained in the non-US countries category, you will see that many users are associated with a country called "N/A." "N/A" is an abbreviation for "Not Applicable"; it is not a real country name. We should separate these entries from the "Outside of the US" category we made earlier. We could use a nested query to say whenever "country" does not equal "US", use the results of a second IF expression to determine whether the outputed value should be "Not Applicable" or "Outside US." The IF expression would look like this:

```
IF(cleaned_users.country='US','In US', IF(cleaned_users.country='N/A','Not Applicable','Outside US'))
```

Since the second IF expression is in the position within the IF expression where you specify "value outputted if conditions are not met," its two possible outputs will only be considered if cleaned_users.country='US' is evaluated as false.

The full query to output the number of unique users in each of the three groups would be:

```
SELECT IF(cleaned_users.country='US','In US',
          IF(cleaned_users.country='N/A','Not Applicable','Outside US')) AS US_user,
       count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
       FROM users
       WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user
```

Try it yourself. You should get 5,642 unique user_guids in the "Not Applicable" category, and 1,263 users in the "Outside US" category.

In [28]:

```
%%sql
SELECT IF(cleaned_users.country='US','In US',
          IF(cleaned_users.country='N/A','Not Applicable','Outside US')) AS US_user,
       count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
       FROM users
       WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.
```

Out[28]:

US_user	count(cleaned_users.user_guid)
In US	9356
Not Applicable	5642
Outside US	1263

The IF function is not supported by all database platforms, and some spell the function as IIF rather than IF, so be sure to double-check how the function works in the platform you are using.

If nested IF expressions seem confusing or hard to read, don't worry, there is a better function available for situations when you want to use conditional logic to output more than two groups. That function is called CASE.

2. CASE expressions

The main purpose of CASE expressions is to return a singular value based on one or more conditional tests. You can think of CASE expressions as an efficient way to write a set of IF and ELSEIF statements. There are two viable syntaxes for CASE expressions. If you need to manipulate values in a current column of your data, you would use this syntax:

```
CASE
 WHEN [condition set 1] THEN [result you want when the conditions in set 1 are met]
 WHEN [condition set 2] THEN [result you want when the conditions in set 2 are met]
 WHEN [condition set 3] THEN [result you want when the conditions in set 3 are met]
 ...(can include as many condition sets as you want)
 ELSE [result you want when none of the condition sets are met]
END
```

Using this syntax, our nested IF statement from above could be written as:

```
SELECT CASE WHEN cleaned_users.country="US" THEN "In US"
            WHEN cleaned_users.country="N/A" THEN "Not Applicable"
            ELSE "Outside US"
            END AS US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user
```

Go ahead and try it:

In [29]:

```
%%sql
SELECT CASE WHEN cleaned_users.country="US" THEN "In US"
            WHEN cleaned_users.country="N/A" THEN "Not Applicable"
            ELSE "Outside US"
            END AS US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.
```

Out[29]:

US_user	count(cleaned_users.user_guid)
In US	9356
Not Applicable	5642
Outside US	1263

Since our query does not require manipulation of any of the values in the country column, though, we could also take advantage of this syntax, which is slightly more compact:

```
CASE column_name or expression
    WHEN [value 1] THEN [result you want when row=value 1]
    WHEN [value 2] THEN [result you want when row=value 2]
    WHEN [value 3] THEN [result you want when row=value 3]
    ... (can include as many values as you want)
    ELSE [result you want when row does not equal any of the specified values]
END
```

Our query written in this syntax would look like this:

```
SELECT CASE cleaned_users.country
            WHEN "US" THEN "In US"
            WHEN "N/A" THEN "Not Applicable"
            ELSE "Outside US"
            END AS US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user
```

Try this query as well:

In [31]:

```
%%sql
SELECT CASE cleaned_users.country
            WHEN "US" THEN "In US"
            WHEN "N/A" THEN "Not Applicable"
            ELSE "Outside US"
            END AS US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.
```

Out[31]:

US_user	count(cleaned_users.user_guid)
In US	9356
Not Applicable	5642
Outside US	1263

In [32]:

```
%%sql
SELECT CASE cleaned_users.country
            WHEN "US" THEN "In US"
            WHEN "N/A" THEN "Not Applicable"
            END AS US_user,
      count(cleaned_users.user_guid)
FROM (SELECT DISTINCT user_guid, country
      FROM users
      WHERE country IS NOT NULL) AS cleaned_users
GROUP BY US_user;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
3 rows affected.
```

Out[32]:

US_user	count(cleaned_users.user_guid)
None	1263
In US	9356
Not Applicable	5642

There are a couple of things to know about CASE expressions:

- Make sure to include the word END at the end of the expression
- CASE expressions do not require parentheses
- ELSE expressions are optional
- If an ELSE expression is omitted, NULL values will be outputted for all rows that do not meet any of the conditions stated explicitly in the expression
- CASE expressions can be used anywhere in a SQL statement, including in GROUP BY, HAVING, and ORDER BY clauses or the SELECT column list.

You will find that CASE statements are useful in many contexts. For example, they can be used to rename or revise values in a column.

Question 3: Write a query using a CASE statement that outputs 3 columns: dog_guid, dog_fixed, and a third column that reads "neutered" every time there is a 1 in the "dog_fixed" column of dogs, "not neutered" every time there is a value of 0 in the "dog_fixed" column of dogs, and "NULL" every time there is a value of anything else in the "dog_fixed" column. Limit your results for troubleshooting purposes.

In [35]:

```
%%sql
SELECT dog_guid, dog_fixed,
CASE dog_fixed
WHEN "1" THEN "neutered"
WHEN "0" THEN "not neutered"
END AS neutered
FROM dogs
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[35]:

dog_guid	dog_fixed	neutered
fd27b272-7144-11e5-ba71-058fbc01cf0b	1	neutered
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	1	neutered
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	0	not neutered
fd27b79a-7144-11e5-ba71-058fbc01cf0b	0	not neutered
fd27b86c-7144-11e5-ba71-058fbc01cf0b	0	not neutered
fd27b948-7144-11e5-ba71-058fbc01cf0b	1	neutered
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	1	neutered
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	1	neutered
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	1	neutered
fd27c5be-7144-11e5-ba71-058fbc01cf0b	1	neutered

CODE

%%sql SELECT dog_guid, dog_fixed, CASE dog_fixed WHEN "1" THEN "neutered" WHEN "0" THEN "not neutered" END AS neutered FROM dogs

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 35050 rows affected.

You can also use CASE statements to standardize or combine several values into one.

Question 4: We learned that NULL values should be treated the same as "0" values in the exclude columns of the dogs and users tables. Write a query using a CASE statement that outputs 3 columns: dog_guid, exclude, and a third column that reads "exclude" every time there is a 1 in the "exclude" column of dogs and "keep" every time there is any other value in the exclude column. Limit your results for troubleshooting purposes.

In [37]:

```
%%sql
SELECT dog_guid, exclude,
CASE exclude
WHEN "1" THEN "exclude"
ELSE "keep"
END AS exclude_cleaned
FROM dogs
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[37]:

dog_guid	exclude	exclude_cleaned
fd27b272-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27b79a-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27b86c-7144-11e5-ba71-058fbc01cf0b	1	exclude
fd27b948-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	1	exclude
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27c5be-7144-11e5-ba71-058fbc01cf0b	None	keep

CODE

%%sql SELECT dog_guid, exclude, CASE exclude WHEN "1" THEN "exclude" ELSE "keep" END AS exclude_cleaned FROM dogs

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 35050 rows affected.

Question 5: Re-write your query from Question 4 using an IF statement instead of a CASE statement.

In [39]:

```
%%sql
SELECT dog_guid, exclude, IF(exclude="1","exclude","keep") AS exclude_cleaned
FROM dogs
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[39]:

dog_guid	exclude	exclude_cleaned
fd27b272-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27b79a-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27b86c-7144-11e5-ba71-058fbc01cf0b	1	exclude
fd27b948-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	1	exclude
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	None	keep
fd27c5be-7144-11e5-ba71-058fbc01cf0b	None	keep

CODE

%%sql SELECT dog_guid, exclude, IF(exclude="1","exclude","keep") AS exclude_cleaned FROM dogs;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 35050 rows affected.

Case expressions are also useful for breaking values in a column up into multiple groups that meet specific criteria or that have specific ranges of values.

Question 6: Write a query that uses a CASE expression to output 3 columns: dog_guid, weight, and a third column that reads...
"very small" when a dog's weight is 1-10 pounds
"small" when a dog's weight is greater than 10 pounds to 30 pounds
"medium" when a dog's weight is greater than 30 pounds to 50 pounds
"large" when a dog's weight is greater than 50 pounds to 85 pounds
"very large" when a dog's weight is greater than 85 pounds
Limit your results for troubleshooting purposes.

Remember that when you use AND to define values between two boundaries, you need to include the variable name in all clauses that define the conditions of the values you want to extract. In other words, you could use this combined clause in your query: "WHEN weight>10 AND weight<=30 THEN "small" …but this combined clause would cause an error: "WHEN weight>10 AND <=30 THEN "small"

In [48]:

```
%%sql
SELECT dog_guid, weight,
   CASE
    WHEN weight<=0 THEN "very small"
    WHEN weight>10 AND weight<=30 THEN "small"
    WHEN weight>30 AND weight<=50 THEN "medium"
    WHEN weight>50 AND weight<=85 THEN "large"
    WHEN weight>85 THEN "very large"
    END AS weight_grouped
FROM dogs
LIMIT 15;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
15 rows affected.
```

Out[48]:

dog_guid	weight	weight_grouped
fd27b272-7144-11e5-ba71-058fbc01cf0b	50	medium
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	20	small
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	70	large
fd27b79a-7144-11e5-ba71-058fbc01cf0b	70	large
fd27b86c-7144-11e5-ba71-058fbc01cf0b	190	very large
fd27b948-7144-11e5-ba71-058fbc01cf0b	60	large
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	190	very large
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	50	medium
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	70	large
fd27c5be-7144-11e5-ba71-058fbc01cf0b	0	very small
fd27c74e-7144-11e5-ba71-058fbc01cf0b	40	medium
fd27c7d0-7144-11e5-ba71-058fbc01cf0b	60	large
fd27c852-7144-11e5-ba71-058fbc01cf0b	20	small
fd27c8d4-7144-11e5-ba71-058fbc01cf0b	50	medium
fd27c956-7144-11e5-ba71-058fbc01cf0b	30	small

CODE

%%sql SELECT dog_guid, weight, CASE WHEN weight<=0 THEN "very small" WHEN weight>10 AND weight<=30 THEN "small" WHEN weight>30 AND weight<=50 THEN "medium" WHEN weight>50 AND weight<=85 THEN "large" WHEN weight>85 THEN "very large" END AS weight_grouped FROM dogs;

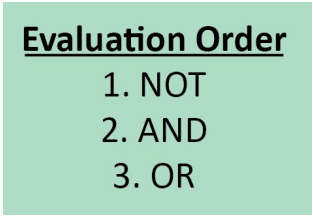
OUTPUT

```
 * mysql://studentuser:***@localhost/dognitiondb
```

35050 rows affected.

3. Pay attention to the order of operations within logical expressions

As you started to see with the query you wrote in Question 6, CASE expressions often end up needing multiple AND and OR operators to accurately describe the logical conditions you want to impose on the groups in your queries. You must pay attention to the order in which these operators are included in your logical expressions, because unless parentheses are included, the NOT operator is always evaluated before an AND operator, and an AND operator is always evaluated before the OR operator.



When parentheses are included, the expressions within the parenthese are evaluated first. That means this expression:

```
CASE WHEN "condition 1" OR "condition 2" AND "condition 3"...
```

will lead to different results than this expression:

```
CASE WHEN "condition 3" AND "condition 1" OR "condition 2"...
```

or this expression:

```
CASE WHEN ("condition 1" OR "condition 2") AND "condition 3"...
```

In the first case you will get rows that meet condition 2 and 3, or condition 1. In the second case you will get rows that meet condition 1 and 3, or condition 2. In the third case, you will get rows that meet condition 1 or 2, and condition 3.

Let's see a concrete example of how the order in which logical operators are evaluated affects query results.

Question 7: How many distinct dog_guids are found in group 1 using this query?

```
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN breed_group='Sporting' OR breed_group='Herding' AND exclude!='1' THEN "group 1"
     ELSE "everything else"
     END AS groups
FROM dogs
GROUP BY groups
```

In [41]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN breed_group='Sporting' OR breed_group='Herding' AND exclude!='1' THEN "group 1"
     ELSE "everything else"
     END AS groups
FROM dogs
GROUP BY groups;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[41]:

COUNT(DISTINCT dog_guid)	groups
30179	everything else
4871	group 1

Question 8: How many distinct dog_guids are found in group 1 using this query?

```
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN exclude!='1' AND breed_group='Sporting' OR breed_group='Herding' THEN "group 1"
     ELSE "everything else"
     END AS group_name
FROM dogs
GROUP BY group_name
```

In [42]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN exclude!='1' AND breed_group='Sporting' OR breed_group='Herding' THEN "group 1"
     ELSE "everything else"
     END AS group_name
FROM dogs
GROUP BY group_name;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[42]:

COUNT(DISTINCT dog_guid)	group_name
31589	everything else
3461	group 1

Question 9: How many distinct dog_guids are found in group 1 using this query?

```
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN exclude!='1' AND (breed_group='Sporting' OR breed_group='Herding') THEN "group 1"
     ELSE "everything else"
     END AS group_name
FROM dogs
GROUP BY group_name
```

In [43]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid),
CASE WHEN exclude!='1' AND (breed_group='Sporting' OR breed_group='Herding') THEN "group 1"
     ELSE "everything else"
     END AS group_name
FROM dogs
GROUP BY group_name;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
2 rows affected.
```

Out[43]:

COUNT(DISTINCT dog_guid)	group_name
35004	everything else
46	group 1

So make sure you always pay attention to the order in which your logical operators are listed in your expressions, and whenever possible, include parentheses to ensure that the expressions are evaluated in the way you intend!

Let's practice some more IF and CASE statements

In order to make it easier to practice SQL queries with meaningful examples before we learned how to join tables, I added extra columns to the "dogs" table that were not in the original Dognition database. These extra columns included the "total_tests_completed" field and multiple inter-test-interval ("iti") summary fields. **Please do NOT try to use these extra fields in the query exercises below. Since you now know how to join tables, we will practice writing queries as if you only had the data provided in the original Dognition database.**

Question 10: For each dog_guid, output its dog_guid, breed_type, number of completed tests, and use an IF statement to include an extra column that reads "Pure_Breed" whenever breed_type equals 'Pure Breed" and "Not_Pure_Breed" whenever breed_type equals anything else. LIMIT your output to 50 rows for troubleshooting. HINT: you will need to use a join to complete this query.

In [45]:

```
%%sql
SELECT d.dog_guid AS dogID, d.breed_type AS breed_type, count(c.created_at) AS
numtests,
IF(d.breed_type='Pure Breed','pure_breed', 'not_pure_breed') AS pure_breed
FROM dogs d, complete_tests c
WHERE d.dog_guid=c.dog_guid
GROUP BY dogID, breed_type, pure_breed
LIMIT 15;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
15 rows affected.
```

Out[45]:

dogID	breed_type	numtests	pure_breed
fd27b272-7144-11e5-ba71-058fbc01cf0b	Pure Breed	21	pure_breed
fd27b5ba-7144-11e5-ba71-058fbc01cf0b	Pure Breed	20	pure_breed
fd27b6b4-7144-11e5-ba71-058fbc01cf0b	Pure Breed	2	pure_breed
fd27b79a-7144-11e5-ba71-058fbc01cf0b	Pure Breed	11	pure_breed
fd27b86c-7144-11e5-ba71-058fbc01cf0b	Pure Breed	31	pure_breed
fd27b948-7144-11e5-ba71-058fbc01cf0b	Pure Breed	20	pure_breed
fd27ba1a-7144-11e5-ba71-058fbc01cf0b	Pure Breed	27	pure_breed
fd27bbbe-7144-11e5-ba71-058fbc01cf0b	Mixed Breed/ Other/ I Don't Know	20	not_pure_breed
fd27c1c2-7144-11e5-ba71-058fbc01cf0b	Pure Breed	20	pure_breed
fd27c5be-7144-11e5-ba71-058fbc01cf0b	Cross Breed	20	not_pure_breed
fd27c74e-7144-11e5-ba71-058fbc01cf0b	Cross Breed	14	not_pure_breed
fd27c7d0-7144-11e5-ba71-058fbc01cf0b	Pure Breed	20	pure_breed
fd27c852-7144-11e5-ba71-058fbc01cf0b	Pure Breed	20	pure_breed
fd27c8d4-7144-11e5-ba71-058fbc01cf0b	Pure Breed	20	pure_breed
fd27c956-7144-11e5-ba71-058fbc01cf0b	Cross Breed	11	not_pure_breed

CODE

%%sql SELECT d.dog_guid AS dogID, d.breed_type AS breed_type, count(c.created_at) AS numtests, IF(d.breed_type='Pure Breed','pure_breed', 'not_pure_breed') AS pure_breed FROM dogs d, complete_tests c WHERE d.dog_guid=c.dog_guid GROUP BY dogID, breed_type, pure_breed;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 17986 rows affected.

**Note that “breed_type” and “pure_breed” are technically optional in this query (above), since dogID should be unique.

Question 11: Write a query that uses a CASE statement to report the number of unique user_guids associated with customers who live in the United States and who are in the following groups of states:

Group 1: New York (abbreviated "NY") or New Jersey (abbreviated "NJ")
Group 2: North Carolina (abbreviated "NC") or South Carolina (abbreviated "SC")
Group 3: California (abbreviated "CA")
Group 4: All other states with non-null values

You should find 898 unique user_guids in Group1.

In [46]:

```
%%sql
SELECT COUNT(DISTINCT user_guid),
CASE
WHEN (state="NY" OR state="NJ") THEN "Group 1-NY/NJ"
WHEN (state="NC" OR state="SC") THEN "Group 2-NC/SC"
WHEN state="CA" THEN "Group 3-CA"
ELSE "Group 4-Other"
END AS state_group
FROM users
WHERE country="US" AND state IS NOT NULL
GROUP BY state_group;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
4 rows affected.
```

Out[46]:

COUNT(DISTINCT user_guid)	state_group
898	Group 1-NY/NJ
653	Group 2-NC/SC
1417	Group 3-CA
6388	Group 4-Other

Question 12: Write a query that allows you to determine how many unique dog_guids are associated with dogs who are DNA tested and have either stargazer or socialite personality dimensions. Your answer should be 70.

In [49]:

```
%%sql
SELECT COUNT(DISTINCT dog_guid)
FROM dogs
WHERE dna_tested=1 AND (dimension='stargazer' OR dimension='socialite');
```

```
 * mysql://studentuser:***@localhost/dognitiondb
1 rows affected.
```

Out[49]:

COUNT(DISTINCT dog_guid)
70

Feel free to practice any other queries you like here!

In [ ]:

```

```

MySQL Exercise 12: Queries that Test Relationships Between Test Completion and Testing Circumstances

In this lesson, we are going to practice integrating more of the concepts we learned over the past few weeks to address whether issues in our Dognition sPAP are related to the number of tests dogs complete. We are going to focus on a subset of the issues listed in the "Features of Testing Circumstances" branch of our sPAP. You will need to look up new functions several times and the final queries at which we will arrive by the end of this lesson will be quite complex, but we will work up to them step-by-step.

To begin, load the sql library and database, and make the Dognition database your default database:

In [5]:

```
%load_ext sql
%sql mysql://studentuser:studentpw@localhost/dognitiondb
%sql USE dognitiondb
```

```
The sql extension is already loaded. To reload it, use:
  %reload_ext sql
 * mysql://studentuser:***@localhost/dognitiondb
0 rows affected.
```

Out[5]:

```
[]
```

In order to make it easier to practice SQL queries with meaningful examples before we learned how to join tables, I added extra columns to the "dogs" table that were not in the original Dognition database. These extra columns included the "total_tests_completed" field and multiple inter-test-interval ("iti") summary fields. **Please do NOT try to use these extra fields in the query exercises below. Since you now know how to join tables, we will practice writing queries as if you only had the data provided in the original Dognition database.**

1. During which weekdays do Dognition users complete the most tests?

The first question we are going to address is whether there is a certain day of the week when users are more or less likely to complete Dognition tests. If so, targeting promotions or reminder emails to those times of the week might increase the number of tests users complete.

At first, the query we need to address this question might seem a bit intimidating, but once you can describe what the query needs to do in words, writing the query won't seem so challenging.

Ultimately, we want a count of the number of tests completed on each day of the week, with all of the dog_guids and user_guids the Dognition team flagged in their exclude column excluded. To achieve this, we are going to have to use the GROUP BY clause to break up counts of the records in the completed_tests table according to days of the week. We will also have to join the completed_tests table with the dogs and users table in order to exclude completed_tests records that are associated with dog_guids or user_guids that should be excluded. First, though, we need a method for extracting the day of the week from a time stamp. In MySQL Exercise 2 we used a function called "DAYNAME". That is the most efficient function to use for this purpose, but not all database systems have this function, so let's try using a different method for the queries in this lesson. Search these sites to find a function that will output a number from 1-7 for time stamps where 1 = Sunday, 2 = Monday, …, 7 = Saturday:

https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html (https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html)
http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

Question 1: Using the function you found in the websites above, write a query that will output one column with the original created_at time stamp from each row in the completed_tests table, and another column with a number that represents the day of the week associated with each of those time stamps. Limit your output to 200 rows starting at row 50.

In [6]:

```
%%sql
SELECT created_at, DAYOFWEEK(created_at)
FROM complete_tests
LIMIT 49,20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[6]:

created_at	DAYOFWEEK(created_at)
2013-02-05 22:10:06	3
2013-02-05 22:23:49	3
2013-02-05 22:26:36	3
2013-02-05 22:29:02	3
2013-02-05 22:32:25	3
2013-02-05 22:33:09	3
2013-02-05 22:36:11	3
2013-02-05 22:38:01	3
2013-02-05 22:48:58	3
2013-02-05 22:53:45	3
2013-02-05 22:59:45	3
2013-02-05 23:01:38	3
2013-02-05 23:04:43	3
2013-02-05 23:06:10	3
2013-02-05 23:35:48	3
2013-02-05 23:40:57	3
2013-02-05 23:45:30	3
2013-02-05 23:48:46	3
2013-02-05 23:54:40	3
2013-02-05 23:59:15	3

Of course, the results of the query in Question 1 would be much easier to interpret if the output included the name of the day of the week (or a relevant abbreviation) associated with each time stamp rather than a number index.

Question 2: Include a CASE statement in the query you wrote in Question 1 to output a third column that provides the weekday name (or an appropriate abbreviation) associated with each created_at time stamp.

In [7]:

```
%%sql
SELECT created_at, DAYOFWEEK(created_at),
       (CASE
        WHEN DAYOFWEEK(created_at)=1 THEN "Su"
        WHEN DAYOFWEEK(created_at)=2 THEN "Mo"
        WHEN DAYOFWEEK(created_at)=3 THEN "Tu"
        WHEN DAYOFWEEK(created_at)=4 THEN "We"
        WHEN DAYOFWEEK(created_at)=5 THEN "Th"
        WHEN DAYOFWEEK(created_at)=6 THEN "Fr"
        WHEN DAYOFWEEK(created_at)=7 THEN "Sa"
        END) AS daylabel
FROM complete_tests
LIMIT 49,20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[7]:

created_at	DAYOFWEEK(created_at)	daylabel
2013-02-05 22:10:06	3	Tu
2013-02-05 22:23:49	3	Tu
2013-02-05 22:26:36	3	Tu
2013-02-05 22:29:02	3	Tu
2013-02-05 22:32:25	3	Tu
2013-02-05 22:33:09	3	Tu
2013-02-05 22:36:11	3	Tu
2013-02-05 22:38:01	3	Tu
2013-02-05 22:48:58	3	Tu
2013-02-05 22:53:45	3	Tu
2013-02-05 22:59:45	3	Tu
2013-02-05 23:01:38	3	Tu
2013-02-05 23:04:43	3	Tu
2013-02-05 23:06:10	3	Tu
2013-02-05 23:35:48	3	Tu
2013-02-05 23:40:57	3	Tu
2013-02-05 23:45:30	3	Tu
2013-02-05 23:48:46	3	Tu
2013-02-05 23:54:40	3	Tu
2013-02-05 23:59:15	3	Tu

Now that we are confident we have the correct syntax for extracting weekday labels from the created_at time stamps, we can start building our larger query that examines the number of tests completed on each weekday.

Question 3: Adapt the query you wrote in Question 2 to report the total number of tests completed on each weekday. Sort the results by the total number of tests completed in descending order. You should get a total of 33,190 tests in the Sunday row of your output.

In [8]:

```
%%sql
SELECT DAYOFWEEK(created_at),COUNT(created_at) AS numtests,
       (CASE
        WHEN DAYOFWEEK(created_at)=1 THEN "Su"
        WHEN DAYOFWEEK(created_at)=2 THEN "Mo"
        WHEN DAYOFWEEK(created_at)=3 THEN "Tu"
        WHEN DAYOFWEEK(created_at)=4 THEN "We"
        WHEN DAYOFWEEK(created_at)=5 THEN "Th"
        WHEN DAYOFWEEK(created_at)=6 THEN "Fr"
        WHEN DAYOFWEEK(created_at)=7 THEN "Sa"
        END) AS daylabel
FROM complete_tests
GROUP BY daylabel
ORDER BY numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

Out[8]:

DAYOFWEEK(created_at)	numtests	daylabel
1	33190	Su
2	30195	Mo
3	27989	Tu
7	27899	Sa
4	26473	We
5	24420	Th
6	23080	Fr

So far these results suggest that users complete the most tests on Sunday night and the fewest tests on Friday night. We need to determine if this trend remains after flagged dog_guids and user_guids are excluded. Let's start by removing the dog_guids that have an exclude flag. We'll exclude user_guids with an exclude flag in later queries.

Question 4: Rewrite the query in Question 3 to exclude the dog_guids that have a value of "1" in the exclude column (Hint: this query will require a join.) This time you should get a total of 31,092 tests in the Sunday row of your output.

In [9]:

```
%%sql
SELECT DAYOFWEEK(c.created_at),COUNT(c.created_at) AS numtests,
        (CASE
         WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
         WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
         WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
         WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
         WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
         WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
         WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
         END) AS daylabel
FROM complete_tests c JOIN dogs d
  ON c.dog_guid=d.dog_guid
WHERE d.exclude IS NULL OR d.exclude=0
GROUP BY daylabel
ORDER BY numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

Out[9]:

DAYOFWEEK(c.created_at)	numtests	daylabel
1	31092	Su
2	28250	Mo
7	26231	Sa
3	25764	Tu
4	24501	We
5	22347	Th
6	21028	Fr

Now we need to exclude the user_guids that have a value of "1" in the exclude column as well. One way to do this would be to join the completed_tests, dogs, and users table with a sequence of inner joins. However, we've seen in previous lessons that there are duplicate rows in the users table. These duplicates will get passed through the join and will affect the count calculations. To illustrate this, compare the following two queries.

Question 5: Write a query to retrieve all the dog_guids for users common to the dogs and users table using the traditional inner join syntax (your output will have 950,331 rows).

CODE

%%sql SELECT dog_guid FROM dogs d INNER JOIN users u ON d.user_guid=u.user_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 950331 rows affected.

In [11]:

```
%%sql
SELECT dog_guid
FROM dogs d INNER JOIN users u
ON d.user_guid=u.user_guid
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[11]:

dog_guid
fd27b272-7144-11e5-ba71-058fbc01cf0b
fd417cac-7144-11e5-ba71-058fbc01cf0b
fd27b5ba-7144-11e5-ba71-058fbc01cf0b
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b
fd27b6b4-7144-11e5-ba71-058fbc01cf0b
fd27b79a-7144-11e5-ba71-058fbc01cf0b
fd27b86c-7144-11e5-ba71-058fbc01cf0b
fd27ba1a-7144-11e5-ba71-058fbc01cf0b
fd27e9a4-7144-11e5-ba71-058fbc01cf0b
fd27ed46-7144-11e5-ba71-058fbc01cf0b
fd3cf718-7144-11e5-ba71-058fbc01cf0b
fd3d587a-7144-11e5-ba71-058fbc01cf0b
fd3fbfe8-7144-11e5-ba71-058fbc01cf0b
fd41c400-7144-11e5-ba71-058fbc01cf0b
fd42e33a-7144-11e5-ba71-058fbc01cf0b
fd3cffe2-7144-11e5-ba71-058fbc01cf0b
fd42e196-7144-11e5-ba71-058fbc01cf0b
fd453b6c-7144-11e5-ba71-058fbc01cf0b
fd43c0c0-7144-11e5-ba71-058fbc01cf0b
fd27b948-7144-11e5-ba71-058fbc01cf0b

Question 6: Write a query to retrieve all the *distinct* dog_guids common to the dogs and users table using the traditional inner join syntax (your output will have 35,048 rows).

CODE

%%sql SELECT DISTINCT dog_guid FROM dogs d JOIN users u ON d.user_guid=u.user_guid;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 35048 rows affected.

In [13]:

```
%%sql
SELECT DISTINCT dog_guid
FROM dogs d JOIN users u
  ON d.user_guid=u.user_guid
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[13]:

dog_guid
fd27b272-7144-11e5-ba71-058fbc01cf0b
fd417cac-7144-11e5-ba71-058fbc01cf0b
fd27b5ba-7144-11e5-ba71-058fbc01cf0b
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b
fd27b6b4-7144-11e5-ba71-058fbc01cf0b
fd27b79a-7144-11e5-ba71-058fbc01cf0b
fd27b86c-7144-11e5-ba71-058fbc01cf0b
fd27ba1a-7144-11e5-ba71-058fbc01cf0b
fd27e9a4-7144-11e5-ba71-058fbc01cf0b
fd27ed46-7144-11e5-ba71-058fbc01cf0b
fd3cf718-7144-11e5-ba71-058fbc01cf0b
fd3d587a-7144-11e5-ba71-058fbc01cf0b
fd3fbfe8-7144-11e5-ba71-058fbc01cf0b
fd41c400-7144-11e5-ba71-058fbc01cf0b
fd42e33a-7144-11e5-ba71-058fbc01cf0b
fd3cffe2-7144-11e5-ba71-058fbc01cf0b
fd42e196-7144-11e5-ba71-058fbc01cf0b
fd453b6c-7144-11e5-ba71-058fbc01cf0b
fd43c0c0-7144-11e5-ba71-058fbc01cf0b
fd27b948-7144-11e5-ba71-058fbc01cf0b

The strategy we will use to handle duplicate rows in the users table will be to, first, write a subquery that retrieves the distinct dog_guids from an inner join between the dogs and users table with the appropriate records excluded. Then, second, we will join the result of this subquery to the complete_tests table and group the results according to the day of the week.

Question 7: Start by writing a query that retrieves distinct dog_guids common to the dogs and users table, excuding dog_guids and user_guids with a "1" in their respective exclude columns (your output will have 34,121 rows).

In [14]:

```
%%sql
SELECT DISTINCT dog_guid
FROM dogs d JOIN users u
ON d.user_guid=u.user_guid
WHERE (u.exclude IS NULL OR u.exclude=0) AND (d.exclude IS NULL OR
d.exclude=0)
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[14]:

dog_guid
fd27b272-7144-11e5-ba71-058fbc01cf0b
fd417cac-7144-11e5-ba71-058fbc01cf0b
fd27b5ba-7144-11e5-ba71-058fbc01cf0b
fd3fb0f2-7144-11e5-ba71-058fbc01cf0b
fd27b6b4-7144-11e5-ba71-058fbc01cf0b
fd27b79a-7144-11e5-ba71-058fbc01cf0b
fd27b948-7144-11e5-ba71-058fbc01cf0b
fd27bbbe-7144-11e5-ba71-058fbc01cf0b
fd27c1c2-7144-11e5-ba71-058fbc01cf0b
fd27c0fa-7144-11e5-ba71-058fbc01cf0b
fd27c5be-7144-11e5-ba71-058fbc01cf0b
fd27c74e-7144-11e5-ba71-058fbc01cf0b
fd27c64a-7144-11e5-ba71-058fbc01cf0b
fd27c7d0-7144-11e5-ba71-058fbc01cf0b
fd27c852-7144-11e5-ba71-058fbc01cf0b
fd27c8d4-7144-11e5-ba71-058fbc01cf0b
fd27cd98-7144-11e5-ba71-058fbc01cf0b
fd27ce1a-7144-11e5-ba71-058fbc01cf0b
fd3d249a-7144-11e5-ba71-058fbc01cf0b
fd27c956-7144-11e5-ba71-058fbc01cf0b

Question 8: Now adapt your query from Question 4 so that it inner joins on the result of the subquery you wrote in Question 7 instead of the dogs table. This will give you a count of the number of tests completed on each day of the week, excluding all of the dog_guids and user_guids that the Dognition team flagged in the exclude columns.

In [15]:

```
%%sql
SELECT DAYOFWEEK(c.created_at) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_at) AS numtests,
         (CASE
          WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
          WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
          WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
          WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
          WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
          WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
          WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
          END) AS daylabel
FROM complete_tests c JOIN (SELECT DISTINCT dog_guid
                            FROM dogs d JOIN users u
                             ON d.user_guid=u.user_guid
                            WHERE ((u.exclude IS NULL OR u.exclude=0)
                            AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
  ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY daylabel
ORDER BY numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
7 rows affected.
```

Out[15]:

dayasnum	year	numtests	daylabel
1	2013	31036	Su
2	2013	28138	Mo
7	2013	26149	Sa
3	2013	25696	Tu
4	2013	24433	We
5	2014	22323	Th
6	2013	21027	Fr

These results still suggest that Sunday is the day when the most tests are completed and Friday is the day when the fewest tests are completed. However, our first query suggested that more tests were completed on Tuesday than Saturday; our current query suggests that slightly more tests are completed on Saturday than Tuesday, now that flagged dog_guids and user_guids are excluded.

It's always a good idea to see if a data pattern replicates before you interpret it too strongly. The ideal way to do this would be to have a completely separate and independent data set to analyze. We don't have such a data set, but we can assess the reliability of the day of the week patterns in a different way. We can test whether the day of the week patterns are the same in all years of our data set.

Question 9: Adapt your query from Question 8 to provide a count of the number of tests completed on each weekday of each year in the Dognition data set. Exclude all dog_guids and user_guids with a value of "1" in their exclude columns. Sort the output by year in ascending order, and then by the total number of tests completed in descending order. HINT: you will need a function described in one of these references to retrieve the year of each time stamp in the created_at field:

https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html (https://dev.mysql.com/doc/refman/5.7/en/sql-function-reference.html)
http://www.w3resource.com/mysql/mysql-functions-and-operators.php (http://www.w3resource.com/mysql/mysql-functions-and-operators.php)

In [16]:

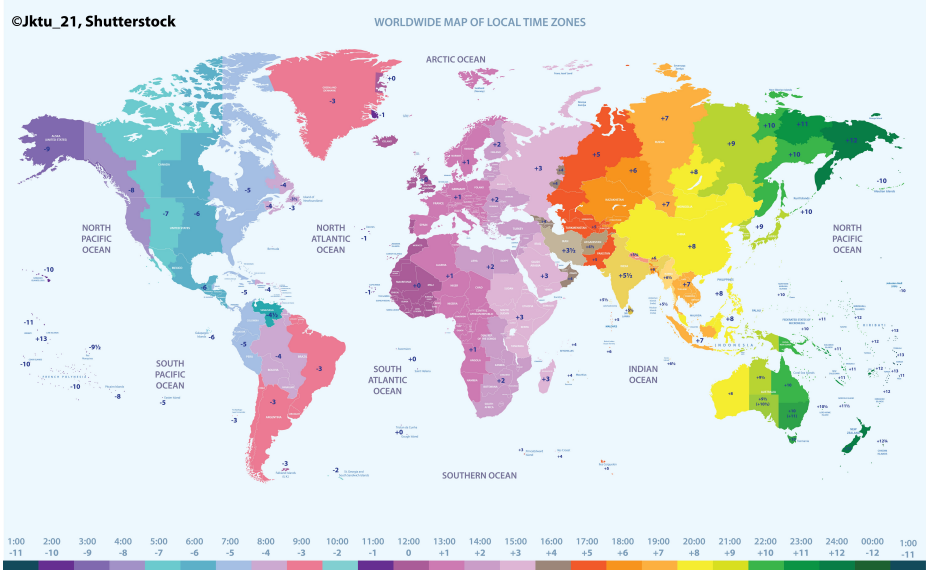
```
%%sql SELECT DAYOFWEEK(c.created_at) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_at) AS numtests,
                  (CASE
                   WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
                   WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
                   WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
                   WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
                   WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
                   WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
                   WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
                   END) AS daylabel
FROM complete_tests c JOIN (SELECT DISTINCT dog_guid
                            FROM dogs d JOIN users u
                              ON d.user_guid=u.user_guid
                            WHERE ((u.exclude IS NULL OR u.exclude=0)
                            AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
  ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY year,daylabel
ORDER BY year ASC, numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[16]:

dayasnum	year	numtests	daylabel
1	2013	8203	Su
7	2013	6854	Sa
2	2013	5740	Mo
4	2013	5665	We
3	2013	5393	Tu
6	2013	4997	Fr
5	2013	4961	Th
2	2014	9309	Mo
1	2014	9210	Su
3	2014	9177	Tu
4	2014	8857	We
7	2014	8257	Sa
5	2014	7286	Th
6	2014	6475	Fr
1	2015	13623	Su
2	2015	13089	Mo
3	2015	11126	Tu
7	2015	11038	Sa
5	2015	10076	Th
4	2015	9911	We
6	2015	9555	Fr

These results suggest that although the precise order of the weekdays with the most to fewest completed tests changes slightly from year to year, Sundays always have a lot of completed tests, and Fridays always have the fewest or close to the fewest completed tests. So far, it seems like it might be a good idea for Dognition to target reminder or encouragement messages to customers on Sundays. However, there is one more issue our analysis does not address. All of the time stamps in the created_at column are in Coordinated Universal Time (abbreviated UTC). This is a time convention that is constant around the globe. Nonetheless, as the picture below illustrates, countries and states have different time zones. The same UTC time can correspond with local times in different countries that are as much as 24 hours apart:



Therefore, the weekdays we have extracted so far may not accurately reflect the weekdays in the local times of different countries. The only way to correct the time stamps for time zone differences is to obtain a table with the time zones of every city, state, or country. Such a table was not available to us in this course, but we can run some analyses that approximate a time zone correction for United States customers.

Question 10: First, adapt your query from Question 9 so that you only examine customers located in the United States, with Hawaii and Alaska residents excluded. HINTS: In this data set, the abbreviation for the United States is "US", the abbreviation for Hawaii is "HI" and the abbreviation for Alaska is "AK". You should have 5,860 tests completed on Sunday of 2013.

In [17]:

```
%%sql
SELECT DAYOFWEEK(c.created_at) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_at) AS numtests,
              (CASE
               WHEN DAYOFWEEK(c.created_at)=1 THEN "Su"
               WHEN DAYOFWEEK(c.created_at)=2 THEN "Mo"
               WHEN DAYOFWEEK(c.created_at)=3 THEN "Tu"
               WHEN DAYOFWEEK(c.created_at)=4 THEN "We"
               WHEN DAYOFWEEK(c.created_at)=5 THEN "Th"
               WHEN DAYOFWEEK(c.created_at)=6 THEN "Fr"
               WHEN DAYOFWEEK(c.created_at)=7 THEN "Sa"
               END) AS daylabel
FROM complete_tests c JOIN
                       (SELECT DISTINCT dog_guid
                        FROM dogs d JOIN users u
                        ON d.user_guid=u.user_guid
                        WHERE ((u.exclude IS NULL OR u.exclude=0)
                        AND u.country="US"
                        AND (u.state!="HI" AND u.state!="AK")
                        AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY year,daylabel
ORDER BY year ASC, numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[17]:

dayasnum	year	numtests	daylabel
1	2013	5860	Su
7	2013	4674	Sa
2	2013	3695	Mo
4	2013	3496	We
3	2013	3449	Tu
6	2013	3163	Fr
5	2013	3090	Th
2	2014	7278	Mo
3	2014	6800	Tu
1	2014	6632	Su
4	2014	6331	We
7	2014	6006	Sa
5	2014	5271	Th
6	2014	4831	Fr
2	2015	8784	Mo
1	2015	8570	Su
3	2015	7218	Tu
7	2015	7116	Sa
5	2015	6343	Th
4	2015	6164	We
6	2015	5689	Fr

The next step is to adjust the created_at times for differences in time zone. Most United States states (excluding Hawaii and Alaska) have a time zone of UTC time -5 hours (in the eastern-most regions) to -8 hours (in the western-most regions). To get a general idea for how much our weekday analysis is likely to change based on time zone, we will subtract 6 hours from every time stamp in the complete_tests table. Although this means our time stamps can be inaccurate by 1 or 2 hours, people are not likely to be playing Dognition games at midnight, so 1-2 hours should not affect the weekdays extracted from each time stamp too much.

The functions used to subtract time differ across database systems, so you should double-check which function you need to use every time you are working with a new database. We will use the date_sub function:

https://www.w3schools.com/sql/func_mysql_date_sub.asp (https://www.w3schools.com/sql/func_mysql_date_sub.asp)

Question 11: Write a query that extracts the original created_at time stamps for rows in the complete_tests table in one column, and the created_at time stamps with 6 hours subtracted in another column. Limit your output to 100 rows.

In [18]:

```
%%sql
SELECT created_at, DATE_SUB(created_at, interval 6 hour) AS corrected_time
FROM complete_tests
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[18]:

created_at	corrected_time
2013-02-05 18:26:54	2013-02-05 12:26:54
2013-02-05 18:31:03	2013-02-05 12:31:03
2013-02-05 18:32:04	2013-02-05 12:32:04
2013-02-05 18:32:25	2013-02-05 12:32:25
2013-02-05 18:32:56	2013-02-05 12:32:56
2013-02-05 18:33:15	2013-02-05 12:33:15
2013-02-05 18:33:33	2013-02-05 12:33:33
2013-02-05 18:33:59	2013-02-05 12:33:59
2013-02-05 18:34:25	2013-02-05 12:34:25
2013-02-05 18:34:39	2013-02-05 12:34:39

Question 12: Use your query from Question 11 to adapt your query from Question 10 in order to provide a count of the number of tests completed on each day of the week, with approximate time zones taken into account, in each year in the Dognition data set. Exclude all dog_guids and user_guids with a value of "1" in their exclude columns. Sort the output by year in ascending order, and then by the total number of tests completed in descending order. HINT: Don't forget to adjust for the time zone in your DAYOFWEEK statement and your CASE statement.

In [19]:

```
%%sql
SELECT DAYOFWEEK(DATE_SUB(created_at, interval 6 hour)) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_
at) AS numtests,
            (CASE
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=1 THEN "Su"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=2 THEN "Mo"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=3 THEN "Tu"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=4 THEN "We"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=5 THEN "Th"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=6 THEN "Fr"
             WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=7 THEN "Sa"
             END) AS daylabel
FROM complete_tests c JOIN
                     (SELECT DISTINCT dog_guid
                      FROM dogs d JOIN users u
                        ON d.user_guid=u.user_guid
                      WHERE ((u.exclude IS NULL OR u.exclude=0)
                      AND u.country="US"
                      AND (u.state!="HI" AND u.state!="AK")
                      AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY year,daylabel
ORDER BY year ASC, numtests DESC;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[19]:

dayasnum	year	numtests	daylabel
1	2013	6061	Su
7	2013	4754	Sa
2	2013	3798	Mo
4	2013	3410	We
3	2013	3276	Tu
5	2013	3079	Th
6	2013	3049	Fr
2	2014	7908	Mo
1	2014	7736	Su
3	2014	6513	Tu
7	2014	6081	Sa
4	2014	5772	We
5	2014	4800	Th
6	2014	4339	Fr
1	2015	10406	Su
2	2015	8229	Mo
7	2015	7154	Sa
3	2015	6673	Tu
4	2015	6266	We
5	2015	5881	Th
6	2015	5275	Fr

You can try re-running the query with time-zone corrections of 5, 7, or 8 hours, and the results remain essentially the same. All of these analyses suggest that customers are most likely to complete tests around Sunday and Monday, and least likely to complete tests around the end of the work week, on Thursday and Friday. This is certainly valuable information for Dognition to take advantage of.

If you were presenting this information to the Dognition team, you might want to present the information in the form of a graph that you make in another program. The graph would be easier to read if the output was ordered according to the days of the week shown in standard calendars, with Monday being the first day and Sunday being the last day. MySQL provides an easy way to do this using the FIELD function in the ORDER BY statement:

https://www.virendrachandak.com/techtalk/mysql-ordering-results-by-specific-field-values/ (https://www.virendrachandak.com/techtalk/mysql-ordering-results-by-specific-field-values/)

Question 13: Adapt your query from Question 12 so that the results are sorted by year in ascending order, and then by the day of the week in the following order: Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday.

In [20]:

```
%%sql
SELECT DAYOFWEEK(DATE_SUB(created_at, interval 6 hour)) AS dayasnum, YEAR(c.created_at) AS year, COUNT(c.created_
at) AS numtests,
              (CASE
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=1 THEN "Su"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=2 THEN "Mo"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=3 THEN "Tu"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=4 THEN "We"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=5 THEN "Th"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=6 THEN "Fr"
               WHEN DAYOFWEEK(DATE_SUB(created_at, interval 6 hour))=7 THEN "Sa"
               END) AS daylabel
FROM complete_tests c JOIN
                    (SELECT DISTINCT dog_guid
                     FROM dogs d JOIN users u
                     ON d.user_guid=u.user_guid
                     WHERE ((u.exclude IS NULL OR u.exclude=0)
                     AND u.country="US"
                     AND (u.state!="HI" AND u.state!="AK")
                     AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY year,daylabel
ORDER BY year ASC, FIELD(daylabel,'Mo','Tu','We','Th','Fr','Sa','Su');
```

```
 * mysql://studentuser:***@localhost/dognitiondb
21 rows affected.
```

Out[20]:

dayasnum	year	numtests	daylabel
2	2013	3798	Mo
3	2013	3276	Tu
4	2013	3410	We
5	2013	3079	Th
6	2013	3049	Fr
7	2013	4754	Sa
1	2013	6061	Su
2	2014	7908	Mo
3	2014	6513	Tu
4	2014	5772	We
5	2014	4800	Th
6	2014	4339	Fr
7	2014	6081	Sa
1	2014	7736	Su
2	2015	8229	Mo
3	2015	6673	Tu
4	2015	6266	We
5	2015	5881	Th
6	2015	5275	Fr
7	2015	7154	Sa
1	2015	10406	Su

Unfortunately other database platforms do not have the ORDER BY FIELD functionality. To achieve the same result in other platforms, you would have to use a CASE statement or a more advanced solution:

http://stackoverflow.com/questions/1309624/simulating-mysqls-order-by-field-in-postgresql (http://stackoverflow.com/questions/1309624/simulating-mysqls-order-by-field-in-postgresql)

The link provided above is to a discussion on stackoverflow.com. Stackoverflow is a great website that, in their words, "is a community of 4.7 million programmers, just like you, helping each other." You can ask questions about SQL queries and get help from other experts, or search through questions posted previously to see if somebody else has already asked a question that is relevant to the problem you are trying to solve. It's a great resource to use whenever you run into trouble with your queries.

2. Which states and countries have the most Dognition users?

You ended up with a pretty long and complex query in the questions above that you tested step-by-step. Many people save these types of queries so that they can be adapted for similar queries in the future without having to redesign and retest the entire query.

In the next two questions, we will practice repurposing previously-designed queries for new questions. Both questions can be answered through relatively minor modifications of the queries you wrote above.

Question 14: Which 5 states within the United States have the most Dognition customers, once all dog_guids and user_guids with a value of "1" in their exclude columns are removed? Try using the following general strategy: count how many unique user_guids are associated with dogs in the complete_tests table, break up the counts according to state, sort the results by counts of unique user_guids in descending order, and then limit your output to 5 rows. California ("CA") and New York ("NY") should be at the top of your list.

CODE

```
%%sql SELECT dogs_cleaned.state AS state, COUNT(DISTINCT dogs_cleaned.user_guid) AS numusers FROM complete_tests c JOIN (SELECT DISTINCT dog_guid, u.user_guid, u.state FROM dogs d JOIN users u ON d.user_guid=u.user_guid WHERE ((u.exclude IS NULL OR u.exclude=0) AND u.country="US" AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned ON c.dog_guid=dogs_cleaned.dog_guid GROUP BY state ORDER BY numusers DESC;
```

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 56 rows affected.

In [23]:

```
%%sql
SELECT dogs_cleaned.state AS state, COUNT(DISTINCT dogs_cleaned.user_guid) AS numusers
FROM complete_tests c JOIN
                     (SELECT DISTINCT dog_guid, u.user_guid, u.state
                      FROM dogs d JOIN users u
                      ON d.user_guid=u.user_guid
                      WHERE ((u.exclude IS NULL OR u.exclude=0)
                      AND u.country="US"
                      AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY state
ORDER BY numusers DESC
LIMIT 20;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
20 rows affected.
```

Out[23]:

state	numusers
CA	1363
NY	628
TX	536
FL	502
NC	467
VA	365
IL	327
PA	321
WA	294
MA	282
OH	270
CO	270
MD	239
NJ	233
GA	218
AZ	211
MI	206
CT	173
OR	153
MN	145

The number of unique Dognition users in California is more than two times greater than any other state. This information could be very helpful to Dognition. Useful follow-up questions would be: were special promotions run in California that weren't run in other states? Did Dognition use advertising channels that are particularly effective in California? If not, what traits differentiate California users from other users? Can these traits be taken advantage of in future marketing efforts or product developments?

Let's try one more analysis that examines testing circumstances from a different angle.

Question 15: Which 10 countries have the most Dognition customers, once all dog_guids and user_guids with a value of "1" in their exclude columns are removed? HINT: don't forget to remove the u.country="US" statement from your WHERE clause.

CODE

%%sql SELECT dogs_cleaned.country AS country, COUNT(DISTINCT dogs_cleaned.user_guid) AS numusers FROM complete_tests c JOIN (SELECT DISTINCT dog_guid, u.user_guid, u.country FROM dogs d JOIN users u ON d.user_guid=u.user_guid WHERE ((u.exclude IS NULL OR u.exclude=0) AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned ON c.dog_guid=dogs_cleaned.dog_guid GROUP BY country ORDER BY numusers DESC;

OUTPUT

- mysql://studentuser:***@localhost/dognitiondb 67 rows affected.

In [25]:

```
%%sql
SELECT dogs_cleaned.country AS country, COUNT(DISTINCT dogs_cleaned.user_guid) AS numusers
FROM complete_tests c JOIN
                   (SELECT DISTINCT dog_guid, u.user_guid, u.country
                    FROM dogs d JOIN users u
                      ON d.user_guid=u.user_guid
                    WHERE ((u.exclude IS NULL OR u.exclude=0)
                    AND (d.exclude IS NULL OR d.exclude=0))) AS dogs_cleaned
    ON c.dog_guid=dogs_cleaned.dog_guid
GROUP BY country
ORDER BY numusers DESC
LIMIT 10;
```

```
 * mysql://studentuser:***@localhost/dognitiondb
10 rows affected.
```

Out[25]:

country	numusers
US	8936
N/A	5466
CA	484
AU	142
GB	123
DE	40
NZ	38
DK	34
NO	30
FR	23

The United States, Canada, Australia, and Great Britain are the countries with the most Dognition users. N/A refers to "not applicable" which essentially means we have no usable country data from those rows. After Great Britain, the number of Dognition users drops quite a lot. This analysis suggests that Dognition is most likely to be used by English-speaking countries. One question Dognition might want to consider is whether there are any countries whose participation would dramatically increase if a translated website were available.

3. Congratulations!

You have now written many complex queries on your own that address real analysis questions about a real business problem. You know how to look up new functions, you know how to troubleshoot your queries by isolating each piece of the query until you are sure the syntax is correct, and you know where to look for help if you get stuck. You are ready to start using SQL in your own business ventures. Keep learning, keep trying new things, and keep asking questions. Congratulations for taking your career to the next level!

There is another video to watch, and of course, more exercises to work through using the Dillard's data set.

In the meantime, enjoy practicing any other queries you want to try here:

In [ ]:

```

```